Matriculation Number:

Exercise 1.

(a) Implement a function `average` that takes a vector of `floats` as argument and returns the average value of the elements in the vector. The vector shall be passed as a const reference, and for an empty vector 0 shall be returned and an error message shall be displayed.

```
#include <iostream>
#include <vector>
using namespace std;

float average(const vector<float> &v)
{
    if(v.size() == 0) {
        cout << "Error: vector is empty!" << endl;
        return 0;
    }

    float sum = 0;
    for(vector<float>::const_iterator it = v.begin();
        it != v.end(); ++it)
        sum += *it;

    return sum / v.size();
}
```

Matriculation Number:

(b) Implement a function sum that takes two vectors of ints as arguments and returns the sum of the two vectors.

If one vector is smaller than the other vector, the "missing" elements in the shorter vector shall be treated as 0 (so the length of the resulting vector is the length of the longer input vector).

Example: (1,2,5,6) + (2,2) = (3,4,5,6)

```
#include <iostream>
#include <vector>
using namespace std;

vector<int> sum(const vector<int> &a, const vector<int> &b)
{
    vector<int>::size_type sz = max(a.size(), b.size());

    vector<int> v(sz);
    for(vector<int>::size_type i = 0; i < sz; ++i) {
        v[i] = 0;
        if(i < a.size()) v[i] += a[i];
        if(i < b.size()) v[i] += b[i];
    }

    return v;
}
```

Matriculation Number:

Exercise 2.

Implement a function `print` which is given a vector of `double`s and an output stream `os` and prints, for each number x contained in the vector `v`, the number of occurrences of x in `v`. Use a map for counting the occurrences.

Example: Given v = (2.1, 4.2, 3.5, 4.2, 2.1, 2.1, 3.5, 2.1, 3.5), the output should be:

```
2.1:   4
3.5:   3
4.0:   2
```

```
#include <ostream>
#include <vector>
#include <map>

using namespace std;

void print(const vector<double> &v, ostream &os) {

    map<double,int> count;

    for(vector<double>::const_iterator it = v.begin();
        it != v.end(); ++it)
        count[*it]++;

    for(map<double,int>::iterator it = count.begin();
        it != count.end(); ++it)
        os << it->first << ":\t" << it->second <<endl;

}
```

Matriculation Number:

Exercise 3.

Implement three classes `Shape`, `Square`, and `Rectangle`, such that `Square` represents a square (with a *width*), `Rectangle` represents a rectangle (with a *width* and a *height*), and `Shape` defines an interface for shapes in general, providing a pure virtual member function `area` for computing the area of a shape.

Design a suitable inheritance hierarchy that expresses: *Squares and rectangles are shapes, and a square is a special kind of a rectangle.* Implement the classes such that each class provides a reasonable constructor and implements the `area` function. Use `double` for representing widths and heights.

```
class Shape {
public:
    virtual double area() const = 0;
};

class Rectangle : public Shape {
    double width, height;

public:
    Rectangle(double w, double h) : width(w), height(h) { }

    double area() const { return width*height; }
};

class Square : public Rectangle {
public:
    Square(double w) : Rectangle(w,w) { }
};
```

Matriculation Number:

Matriculation Number:

Exercise 4.

(a) Read and understand the following program:

```
#include <iostream>

using namespace std;

int main()
{
  short x, y, z;
  short *p1, *p2, *p3;

  p1 = &x;  p2 = &y;  p3 = &z;
  *p1 = 20;  *p2 = 1;  *p3 = 15;
  cout << "x=" << x << ", y=" << y << ", z=" << z << endl; // CHECK1

  p2 = p1; p1 = &z;  p3 = &y;
  *p1 = 4; *p3 = *p3 - 1;
  cout << "x=" << x << ", y=" << y << ", z=" << z << endl; // CHECK2

  *p3 = *p1 + *p2;
  p1 = p2;
  *p1 = *p1 + 10;  *p2 = *p2 - 1;
  cout << "x=" << x << ", y=" << y << ", z=" << z << endl; // CHECK3

  return 0;
}
```

Fill the table with the values of the variables x, y, and z at the three checkpoints.

	x	y	z
CHECK1	20	1	15
CHECK2	20	0	4
CHECK3	29	24	4

(b) Read and understand the following program:

```
#include <iostream>
using namespace std;

class A {
  static int counter;
  int c;
public:
  A() { c = ++counter; }
  int get_c() { return c; }

  void id() { cout << "class A" << endl; }
  virtual void mycount() { cout << "A: " << c << endl; }
};

int A::counter;

class B : public A {
public:
  void id() { cout << "class B" << endl; }
  void mycount() { cout << "B: " << get_c() << endl; }
};

int main() {
  A a; B b;
  A &ar = a; A &br = b;

  ar.id(); ar.mycount();
  br.id(); br.mycount();

  return 0;
}
```

What is the output of the program?

```
class A
A: 1
class A
B: 2
```

Matriculation Number:

Room for Notes

Matriculation Number:

Room for Notes

Room for Notes