Question 1

```
/** This function check if an int array is ordered.
    warning : empty array is considered as ordered. */
bool scoresIncreasing(int scores[], int nb_score) {
    if(nb_score > 0 && scores != NULL) {
        for(int i=1; i<nb_score; i++) {
            if(scores[i-1] > scores[i])
                return false;
        }
    }
    return true;
}
```

Question 2

```
bool hasOne(int val) {
    while(val != 0) {
        if(val%10 == 1)
            return true;
        val /= 10;
    }
    return false;
}
```

Question 3

```
class Vehicle {

    protected:
        string color;
        string make;
        string model;

    public:
        Vehicle(string color_ = "unknow_color",
                string make_ = "unknow_manufacter",
                string model_ = "unknow_model") : color(color_),
                                                  make(make_),
                                                  model(model_) {};

        virtual void display() {
            cout << "This vehicle is a " << color << " " << model
                 << " made by " << make << "." << endl;
        };
};
```

Question 4

```
class Car : public Vehicle {

    private:
        int nb_seats;

    public:
        void display() {
            Vehicle::display();
            cout << "Furthermore, this is a car which have "
                 << nb_seats << " seat(s)." << endl;
        };

        Car(string color_ = "unknow_color",
            string make_ = "unknow_manufacter",
            string model_ = "unknow_model",
            int nb_seats_ = 4) : Vehicle(color_, make_, model_),
                                 nb_seats(nb_seats_) {};
};
```

Question 5

```
class Vehicle {

    protected:
        int id;
        string color;
        string make;
        string model;

    public:
        static int s_current_id;

        Vehicle(string color_ = "unknow_color",
                string make_ = "unknow_manufacter",
                string model_ = "unknow_model") : color(color_),
                                                  make(make_),
                                                  model(model_) {
            id = s_current_id;
            s_current_id++;
        };

        virtual void display() {
            cout << "This vehicle is a " << color << " " << model
                 << " made by " << make << "." << endl
                 << "His number id is " << id << "." << endl;
        };
};

int Vehicle::s_current_id = 1000;
```

Question 6

```
Vehicle(string color_ = "unknow_color",
        string make_ = "unknow_manufacter",
        string model_ = "unknow_model") : color(color_),
                                          make(make_),
                                          model(model_) {
      id = s_current_id;
      s_current_id++;
      cout << "vehicle created" << endl;
};

~Vehicle() {
      cout << "vehicle destroyed" << endl;
};

Car(string color_ = "unknow_color",
    string make_ = "unknow_manufacter",
    string model_ = "unknow_model",
    int nb_seats_ = 4) : Vehicle(color_, make_, model_),
                         nb_seats(nb_seats_) {
      cout << "car created" << endl;
};

~Car() {
      cout << "car destroyed" << endl;
};

int main()
{
    Car my_car;
    return 0;
}
```

Results :

1. vehicle created
2. car created
3. car destroyed
4. vehicle destroyed

This result is logic because the Car class inherit from Vehicle. Thus, a Vehicle need to be created before we can add the "Car-specific" part.

Question 7

```
int main()
{
    Car* my_car = new Car();
    delete my_car;
    return 0;
}
```

Results : same as Question 6.

Question 8

```
inline string getColor() {
    return color;
}

inline void setColor(string new_color) {
    color = new_color;
}
```

Question 9

```
int factorial(int val) {
    if(val > 1)
        return val*factorial(val-1);
    else return 1;
}
```

Question 10

```
#include <iostream>
#include <cstring>

using namespace std;

int main(int argc, char *argv[])
{
    int freq_occurences[10] = {0};
    for(int i=1; i<argc; i++) {
        if( strlen(argv[i]-1) <= 10 && strlen(argv[i]-1) > 0 ) {
            freq_occurences[strlen(argv[i])-1]++;
        }
        else {
            cout << "one string size is out of range" << endl;
        }
    }

    cout << "there are " << argc -1 << " strings" << endl;
    cout << "The number of strings with: " << endl;
    for(int i=0; i<10; i++)
        cout << "Length " << i+1 << " characters: "
             << freq_occurences[i] << endl;
}
```