**Ministry of education and science of the Kyrgyz Republic**

**Kyrgyz State Technical University named after I.Razzakov**

**Faculty of Information Technologies**

**Department of Software of Computer Systems**

**Major: 710400 «Software Engineering»**

Report

Discipline: «**Object-Oriented Design**»

Software requirements

Done by**:** student of the SE(eng)- 1- 21

Rami Alshalabi

Checked by: Musabaev E.B.

Bishkek 2024

# Software Requirements Specification (SRS)

The SRS should detail the functional and non-functional requirements, such as:

* Functional:
  + The system must compare the speeds of Kamaz and Tatra trucks on different terrains.
  + The system must declare a winner based on overall performance across all terrains.
  + The system should provide a user-friendly interface to display results.
* Non-functional:
  + The system should ensure extensibility for adding more trucks or terrains.
  + The system should be robust and handle invalid inputs gracefully.

Design

* **Input Interface**: Prompt the user to enter speed values for Kamaz and Tatra on plains, mountains, and deserts.
* **Output Interface**: Display a table of results showing each truck's speed on different terrains and the overall winner.

# Structure and Development

## The program should include:

* + Class definitions for Kamaz and Tatra.
  + Implementation of the **FrC\_greater()** function.
  + Main function to handle input, perform comparisons, and output results.

## Program Block Diagram

1. Input speeds for each terrain.
2. Compare speeds using **FrC\_greater()**.
3. Calculate and display the overall performance.
4. Declare the winner.

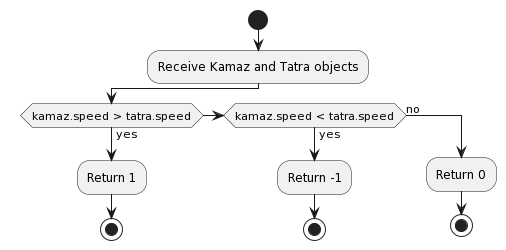


Figure 1. FrC\_greater()

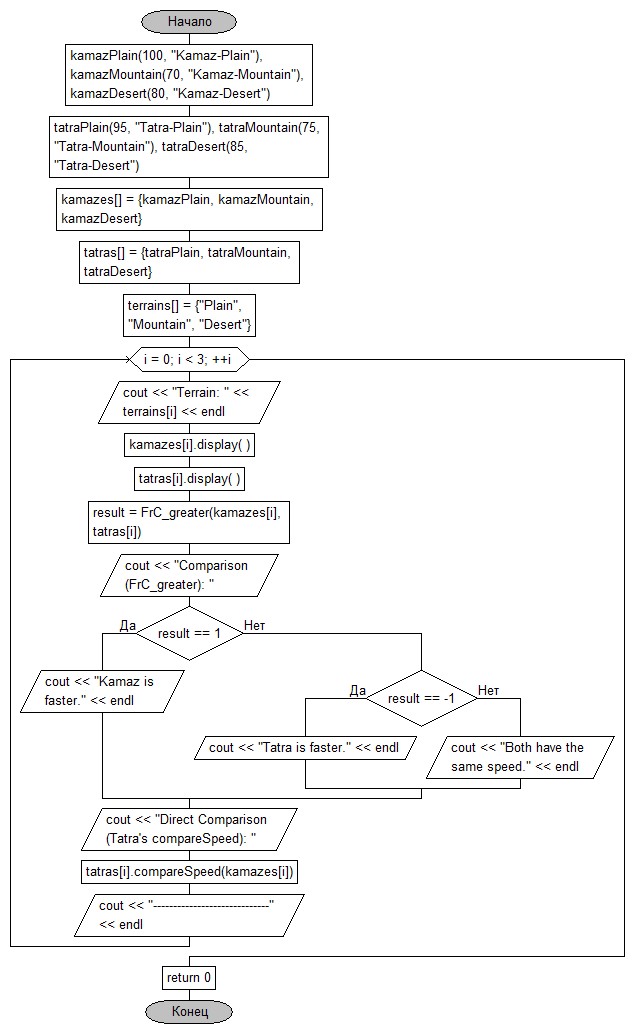


Figure 2. BD for the main demonstration.

# QUESTIONS FOR THE SUBMISSION OF LABORATORY WORK No. 4

### 1. What is a friendly function, what is it for?

A friendly function is a function that is not a member of a class but has access to its private and protected members. Friend functions are used to perform operations on class objects that require direct access to the class's internal members without being part of the class itself.

### 2. How is it declared, what is its syntax?

A friend function is declared inside the class definition with the **friend** keyword preceding its prototype. The syntax is as follows:

class ClassName { friend ReturnType FunctionName(Parameters); };

### 3. How can a friend function access a private class field?

A friend function can access private (and protected) fields of a class because the class declaration explicitly grants access by declaring the function as a friend. This bypasses the usual encapsulation restrictions.

### 4. Which option for calling DF is correct: a.show() or show(a)?

For a friend function, the correct syntax is **show(a)**, where **show** is the friend function and **a** is the object being passed to it. **a.show()** implies that **show** is a member function of the object **a**, which is not the case for friend functions.

### 5. Is a friend function inherited?

No, friend functions are not inherited because they are not members of the class to begin with. Each derived class must separately declare functions as friends if needed.

### 6. Can a friend function be friendly to more than one class?

Yes, a friend function can be declared as a friend in multiple class definitions, allowing it to access private and protected members of all those classes.

### 7. For what problems does it make sense to use a function that is friendly to two or more classes?

Using a friend function makes sense in scenarios where a function needs to access and manipulate the private data of multiple classes, such as operator overloading that involves objects of different classes, or when implementing external functions that need detailed access to the internals of several classes to perform their tasks efficiently.

### 8. When should you use friend classes?

Friend classes should be used when one class needs extensive access to the private and protected members of another class, but inheritance or public access methods are not appropriate or would violate encapsulation principles. It's particularly useful in tightly coupled classes that are designed to work closely together.

### 9. Which OOP principle does friend functions not follow?

Friend functions do not follow the encapsulation principle strictly. Encapsulation is about hiding the internal state of an object and requiring all interactions to be performed through an object's methods. By allowing external functions direct access to an object's internal state, this principle is somewhat compromised.

### 10. How can you limit the use of DF?

To limit the use of friend functions, they should be used sparingly and only when absolutely necessary. Additionally, limiting the number of classes that declare a particular function as a friend and keeping the scope of friend functions focused can help minimize their impact on encapsulation.

### 11. In what extreme cases should DF be used?

Friend functions should be used in extreme cases where an external function needs to access the private members of a class to perform operations that cannot be accomplished via the class's public interface, and where using member functions or friend classes is impractical. This includes complex operations involving multiple classes, such as mathematical operations or comparison functions that require direct access to the internals of the classes involved.