This scheme is from CM course: <https://bb.cranfield.ac.uk/bbcswebdav/pid-674563-dt-content-rid-1099260_2/courses/N-CST-CM-18-A18/Report-Format%282%29.pdf>

*We can remember about important rules in documentation.*

**Title**

**Contents**

**List of Figures**

**List of Tables**

**List of Abbreviations**

**Abstract**

* Self-explanatory without reference to the paper.
* Max 2 paragraphs, NO ABBREVIATIONS HERE.
* Concisely indicate the experiment, objectives, importance.
* Newly observed facts and the conclusions of the experiment.
* Only the most significant results should be given.

**Introduction**

* Statement of the problem investigated.
* Background information.
* Concise introduction to theory or concepts used.
* Subsections.

**Methods/Procedures**

* The experimental design, methods of gathering data.
* Subsections.

**Results**

* Heart of the report.
* Actual results, implications, errors.
* Combine Results and Discussions when the discussion of first result is needed to
* understand second result.
* Tables, figures, Graphs APPROPRIATELY REFERENCED IN THE TEXT.
* Figure legends -- below the figure; table legends -- above the table.

**Discussion**

* Interpretation
* Any patterns/relationships/correlations that were observed that were important.
* Future work
* If the results are not definitive, specific future work that may be needed can be
* (briefly) described.

**Conclusions**

* What the author thinks the results mean based on the observations.
* Should relate directly back to the problem/question stated in the introduction.
* DO NOT INTRODUCE SURPRISES HERE.

**Acknowledgments**

**References**

**Appendices**

* Pseudocode
* Program listing
* Information that is too detailed to be placed into the report's text.
* Information that does not directly concern the experiment's objectives.

--------------------------------------------------------------------------------------------------------------------------------------

**Questions from assignment document about OOP:**

1. What classes (abstractions) will you use?
2. What will be the separation of responsibilities into the chosen classes i.e what is each class going to be responsible for doing?
3. What will be the data/methods encapsulation for each class?
4. What are the relationships between the chosen classes: aggregation (‘has a’ or ‘contains’), inheritance (‘is a kind of’), association (‘uses’ - one class may use objects of another class in the implementation of its methods)?
5. Try to adhere to the SOLID principles for effective design (see BB pages for C++).
6. Try to use the Standard Library components where appropriate (containers, iterators, algorithms, numerics). This is well designed, thoroughly tested, robust and efficient code.
7. Try to use exceptions to deal with exceptional conditions and catch them at an appropriate higher level.
8. Try not to overcomplicate things. Keep things simple but effective.

**Answers:**

1. In Math, different schemes are used to compute different equation. However, they have common properties, e.g. the same technique to compute initial and boundary conditions. Of course, these schemes have various way to solve a problem. Thereby, in application has been implemented three important classes like: Scheme, which is main, abstract class, ImplicitScheme and ExplicitScheme class, which inherit by Scheme. Additionally, programme includes Matrix and Vector classes, which are helpful to store information about initial condition of equation or final result.
2. The Scheme class contains elements, which are common for every type of scheme available in the exercise. Inherited classes have access to these elements (methods, variables), so they can be implemented only once in mother class. The Scheme is responsible for compute initial and analytical solution, which are the same for every type of scheme (in this exercise). Additionally, this class include method, which determines size of matrix and value of ∆T. The Scheme contains also method for analytical solution and other methods, which are helpful to represent results in console or in files.

Classes, which derives from Scheme, contain implementation of each implicit and explicit schemes. These methods are unique, so the best way was to put them into appropriate classes.

The Matrix class is responsible to storing values in containers, printing it. In that, programme is more readable and easier to understand.

1. The most important information is some part of class shall not be public. In our programme, three known types of access specifiers are used – public, private and protected. Data members of Scheme class are protected. They can be used only in methods of this class and class derives from Scheme. Other classes shall have access to this data members by public methods – accessors. The Scheme class includes also protected methods, which are common for every type of scheme like boundary or initial condition, compute delta t and size of matrix. They are used only inside of class that derive from Scheme class. Additionally, other public methods exist in Scheme class.

Classes that derived from Scheme class include public methods. These methods can be called in main function.

Data members and methods in Vector and Matrix classes are always public, because different classes need access to containers and use it. This solution is not dangerous, if we talk about security of application.

1. Aggregation has been implemented in Scheme class. Type of Matrix has been used inside of Scheme class as data member (matrix, analytical solution) and as method (GetMatrix).

As mentioned above, inheritance has been implemented in case of Scheme, ExplicitScheme and ImplicitScheme classes. Additionally, Scheme class is virtual, so this is virtual inheritance.

At the end, it is important to mention that association does not exist in our implementation. However, keywords like ‘friend’ or ‘const’ have been used.

1. Our code qualifies some SOLID principles.
   1. Single Responsibility Principle

Every class has only one responsibility.

* 1. Open closed principle

If implementation of core class has been extended to new method, other modules have not been modified.

* 1. Liskov's Substitution Principle

Inheritance is used properly.

* 1. The Interface Segregation Principle

This principle is not very noticeable in our implementation of code. Maybe better way shall be implement additional abstract class (interface) like SchemeUI, which can contain method like Print and Save. Now, these methods are in one abstract class Scheme.

* 1. The Dependency Inversion Principle

If any method has been changed in low level code like in ExplicitScheme and ImplicitScheme classes, any part of code has not been fixed in how level code like in Scheme class (which is core class).

1. STL is used in our implementation.
   1. Special containers like vector has been implemented in Matrix and Vector class. The big advantages of this type of container are short insertion time (at the end) and access like in array. Additionally, creating matrix from vectors is possible. Vector has a few disadvantages, but it is not important in our case. Vectors are not resizing during the working of program, any value is not searching and only insertion at the end is needed.
   2. The other type of element from STL library has been used in our code, especially in ImplicitScheme class. This is the iterator, which can reduce the complexity and execution time of program.
2. *A few information about exceptions in c++ and in our code… (Exceptions will be added at the end of our work!)*
3. Program for computing equation in different ways has been implemented as simple as possible, but effective.
   1. Vectors and iterators are important part in effectiveness.
   2. Classes enable to division of responsibilities.
   3. Code is not repeatable. Each method has unique implementation.
   4. Some elements of code like keyword ‘friend’ has been added to practise object-oriented programming in C++.

Other stuff, which will be added to report.

1. Documentation for classes ->doxygen.
2. Code with comment in Appendix.