1 Introduction

During this practical assignment you will be required to implement the Template Method and Memento design patterns. The assignment consists of two independent tasks, one for each design pattern.

After successful completion of this assignment you should be comfortable with the following concepts:

- Implementing classes from a UML class diagram;
- Understanding
  - how certain steps in the Template Method can be deferred to subclasses;
  - how polymorphism can be used with the Template Method;
  - how to store a state using the Memento;
  - how the ownership of the Memento is transferred from the Originator to the Caretaker;
  - how probabilities can be simulated with random number generation.

2 Constraints

1. You must complete this assignment individually.
2. You may ask the Teaching Assistants for help but they will not be allowed to give you the solutions.

3 Submission Instructions

You are required to upload all your source files as a single tar.gz archive to the CS website before the deadline. You are required to implement all makefiles, headers and source files for both tasks yourself from the given UML class diagrams. You can use the provided Main.cpp to test your code. The uploaded archive should have two directories named Task1_TemplateMethod and Task2_Memento each containing all the sources and makefile for the corresponding task.

4 Mark Allocation

<table>
<thead>
<tr>
<th>Task</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template Method - Plotting Mathematical Functions</td>
<td>20</td>
</tr>
<tr>
<td>Memento - Pokémon Game</td>
<td>25</td>
</tr>
<tr>
<td>TOTAL</td>
<td>45</td>
</tr>
</tbody>
</table>
5 Assignment Instructions

This practical assignment consists of two independent tasks. The first task requires the implementation of three different mathematical functions using the Template Method design pattern. The second task is a Pokémon game that should be implemented with the Memento design pattern.

Task 1: Template Method - Plotting Mathematical Functions .................. (20 marks)

It is very common in a mathematical context to make use of mathematical functions to draw a visual representation of it in terms of a graph. This task should illustrate how the Template Method can be used to plug in different behaviour for similar problems.

You have already received a Plotter class which has been partially implemented. Complete this class’s constructor to initialise the default MathematicalFunction as the Sigmoid function (more details below), the destructor for memory management and the MathematicalFunction’s setter.

- The Sigmoid function $f(x) = \frac{a}{1 + e^{-x}}$ as f in the Sigmoid class
- The Sine function $f(x) = a \sin(x)$ as f in the Sin class
- A straight line $f(x) = ax$ as f in the StraightLine
- Any other mathematical function which is displayable by Plotter Such as a hyperbola etc.

You must also create the following MathematicalFunctions as concrete realizations of the abstract interface called MathematicalFunction.

The MathematicalFunction interface must be as follows, where amp is the amplitude (or gradient) of the function:

```cpp
public:
    virtual int f(int x, double amp) = 0;
```

TIP: Remember to take integer operations into account when doing your math. You may use the cmath library.

Test your program with the Main provided. Your output should look as is shown on the next page.

You are allowed to adjust the width and height setting in Main to accommodate for different resolutions and screen sizes, your output must however display the visual representation of the function fully.
Task 2: Memento - Pokémon Game ......................................................... (25 marks)

For this task you have to implement a small Pokémon game using the Memento design pattern. The logic and game flow was already implemented for you and can be found in Main.cpp. Pokémon is a Nintendo franchise around the world of pocket monsters that can be trained and evolved. You are required to implement the Memento design pattern where Pokémon starts at an evolution level of 1. The Pokémon can then be evolved by increasing its evolution level. During the game the user can decide to save the current state of the Pokémon. At a later stage the previous evolution levels can be loaded again, which devolves your Pokémon. As you can see in the UML class diagram below, there is an additional aggregation between Pokemon and Trainer which is not part of the Memento design pattern, but is shown since the trainer will create, train and take care of
The Pokémon.

The Originator class Pokemon has the following members:

- **mName:** A string variable holding the name of the Pokémon.
- **mEvolutionLevel:** An integer variable holding the current evolution level of the Pokémon.
- **Constructor:** The constructor should initialize the Pokémon name, set the initial evolution level to 1 and print out a statement with the name indicating that the Pokémon was created.
- **getName:** A function that returns the Pokémon name.
- **evolve:** A function that increments the evolution level of the Pokémon by 1 and prints out a statement with the Pokémon name and the new level.
- **saveEvolution:** A function that dynamically allocates a new Memento object with the current evolution level. The function should also print a statement with the Pokémon name and its level, indicating that the evolution was saved.
- **loadEvolution:** A function takes a Memento pointer as parameter and changes the Pokémon’s evolution level to the level previously saved in the Memento. The function should also print a statement with the Pokémon name and the new evolution level, indicating that the evolution was loaded.

The Memento class Evolution has the following members:

- **mLevel:** An integer variable holding the evolution level of the Pokémon that will be saved for later retrieval.
- **Constructor:** The constructor takes the evolution level as a parameter and initializes its member variable. Note that this Memento does not have a function to set the state, this can only be done through the constructor.
- **getLevel:** A function that returns the evolution level.

The Caretaker class Trainer has the following members:

- **mPokemon:** A pointer to a Pokemon object.
- **mEvolutions:** An array of Evolution pointers. These Memento objects will be used to load a previously saved level.
- **mMaximumSaves:** An integer variable indicating the maximum number of saves you want to allow. This will also be the size of the array mEvolutions.
- **mIndex:** An integer variable holding the current number of saves.
- **Constructor:** The constructor takes a Pokémon name as parameter and should allocate a Pokemon object. The second parameter holds the maximum number of saves and should be used to allocate the array mEvolutions. Remember to initialize mIndex (no Mementos were saved yet).
- **Destructor:** The destructor should be used to deallocate the memory for all dynamic allocations. Since mEvolutions is an array of pointers, you first have to delete all pointers in the array before deallocating the array itself.
- **fight:** This function should be called to let your Pokémon fight against another one. Your Pokémon has a 30% chance to win the battle. If your Pokémon wins, it should evolve and a message should be printed, indicating the Pokémon’s name and that it won. If it looses, the Pokémon should not evolve...
and a message with the name should be printed, indicating that it lost. You can simulate probabilities by using random number generation.

- **save**: This function should be used to save the trainer’s Pokémon evolution state. The function should check if there is still an open space in the `mEvolutions` array. If the array is full, -1 should be returned. If the array is not full yet, the `Memento` created by the `Originator` should be added to the array and the position of the newly added `Memento` in the array should be returned. Remember to update `mIndex`.

- **load**: This function should load a previously saved evolution level of the trainer’s Pokémon. The parameter is the index of the `Memento` in the member array. If the requested position is invalid, `false` should be returned. If the position is valid, the Pokémon should be devolved with the `Memento` in the array at the given position and `true` should be returned.