BLG 252E - Object Oriented Programming
Assignment #2

Due: April, 27th 23:59

Introduction

In this assignment, you are expected to implement a simplified version of a simulator for a computer. Computer entities would execute operations using CPU (Central Processing Unit) and GPU (Graphics Processing Unit) entities which use ALU (Arithmetic Logic Unit) and CUDA cores, respectively. This assignment aims to support the understanding of the relationships between classes.

For any issues regarding the assignment, please contact Erhan Biçer (**bicer21@itu.edu.tr**).

Implementation Notes

The implementation details below are included in the grading:

1. Please follow a consistent coding style (indentation, variable names, etc.) with comments.
2. Do not use any pre-compiled header files or STL commands.
3. You may (and should) implement any getter and setter methods.
4. Make sure that there is **no memory leak** in your code.
5. Define required methods and variables as **const** in all necessary cases.
6. Do not make any modifications in the given main.cpp. Your code will be evaluated using the given main.cpp.
7. Ensure that your outputs match with the sample scenario for given inputs. You will be provided with a calico file to check your assignment output.

Submission Notes

1. Your program should compile and run on Linux environment. You should code and test your program on Docker's Virtual Environment.
2. As source code submission, send only **Computer.h** and **Computer.cpp** that contain all necessary declarations and definitions. Along with source codes, you should submit **report** and **log file**.
3. Do not change included libraries and path to header files.
4. Submissions are made through **only** the Ninova system and have a strict deadline. Assignments submitted after the deadline will not be accepted.
5. This is not a group assignment, and **getting involved in any kind of cheating is subject to disciplinary actions**. Your homework should not include any copy-paste material (from the Internet or from someone else's paper/thesis/project).

Operations	Unit
add	CPU
subtract	CPU
multiply	CPU
render	GPU
trainModel	GPU

Table 1: Operations and responsible components

1 Implementation Details

You are expected to implement five classes; **Computer**, **CPU**, **GPU**, **ALU**, and **CUDA**.

1.1 Computer

The computer can perform five operations using its components, as given in Table 1.

1. Computer class has two attributes, **attachedCPU** and **attachedGPU** that point to attached CPU and GPU, respectively. The computer never modifies the internal states of the attached CPU and GPU.
2. CPU and GPU can be attached to a computer using the "+" operator. Operands should be a **computer class object** and **reference to CPU or GPU class object**. **Adding new entities to a computer is not possible if other entities are already attached.**
3. To execute an operation on the computer, **execute** method is called by giving the operation name as an argument. Defined operation types and the responsible unit are stated in Table 1. The computer sends the operation as an argument to the **relevant component**'s execute method according to the given table and **prints the result**. Execution does not change the attributes of the CPU and GPU.

1.2 CPU

1. CPU class contains an ALU object. CPU initializes ALU's **numPerCores** attribute.
2. A CPU can execute relevant operations with **execute** method. **It reads two integers from the keyboard, delegates the operation to the ALU, and returns the result.**

1.3 GPU

1. A GPU contains a CUDA object. GPU initializes CUDA's **numCores** variable.
2. A GPU can execute operations with **execute** method. Within this method, it uses its **cuda** for operations and **returns the result.**

1.4 ALU

1. ALU contains an integer **numPerCores** variable representing the number of ALUs per CPU core.
2. ALU contains **add**, **subtract** and **multiply** methods. These methods take two **integers** as arguments and returns the result.

1.5 CUDA

1. CUDA contains an integer **numCores** variable, representing the number of CUDA cores.
2. CUDA cores can be used to render a video by calling its **render** method. This method will return the result as: "Video is rendered".
3. CUDA cores can be used to train an AI model by calling its **trainModel** method. This method will return the result as: "AI Model is trained".

Notice: Member variables should not be accessed directly outside of the class.

2 Main method

Your output should be like the below after running the given main.cpp:

Command Line

```
Computer is ready
ALU is ready
CPU is ready
CUDA is ready
GPU is ready
CPU is attached
GPU is attached
ALU is ready
CPU is ready
CUDA is ready
GPU is ready
There is already a CPU
There is already a GPU
Operation type is:
subtract
Enter two integers
5
3
2
Operation type is:
render
Video is rendered
Operation type is:
trainModel
AI Model is trained
Operation type is:
add
Enter two integers
1
2
3
Operation type is:
multiply
Enter two integers
4
5
20
```

3 Report

Draw a **UML class diagram** that represents the **association** between the classes. **Show special type of associations if there is any.** You can use this website to draw a UML diagram. Below the diagram, **write down all of the associations**. Also, write **examples of parameter visibility if there is any**.

4 How to compile, run and test your code:

You can build and run your code with the below commands. Otherwise, you can work with VSCode for debugging and running the code. In VSCode, **ensure that the assignment folder is opened in the explorer.**

```
$ g++ -Wall -Werror src/Computer.cpp src/main.cpp -I include -o assignment2
$ ./assignment2
```

Given yaml file checks both for memory leak using Valgrind and test the outputs using calico. To check your code, run the below command and **send the log through Ninova**:

```
$ calico assignment2_eval.yaml --debug |& tee calico_log.txt
```