C++ Advanced – Exam 1 (29 Feb 2020)

Write C++ code for solving the tasks on the following pages.

Code should compile under the C++11 standard.

Submit your solutions here: https://judge.softuni.bg/Contests/1805/

Any code files that are part of the task are provided under the folder **Skeleton**.

Please follow the exact instructions on uploading the solutions for each task.

Task 3 – Memory Pool

Your task is to write a program that represents a famous programming design pattern – "**Memory Pool**".
Because dynamic memory allocation(call to the **new** operator) is a relatively costly operation – frequent dynamic memory may harm your application performance.
The **MemoryPool** design pattern addresses this issue by making one big allocation in the beginning of your program and then only "borrows" memory when you need it. After you are done using the memory you "return" it back to the **MemoryPool** (but no calling to **delete** operator).
When the MemoryPool instance is destroyed only then the memory is actually freed.

You are given a header "**MemoryPool.h**"**.**
Your task is to **provide an implementation** for the MemoryPool class functionalities (for example in a MemoryPool.cpp file).

You are given the **`main()`** function reads 2 integers (**N** and **M**).
M is the **size** of the memory pool underlying memory.
On the next **N** rows different commands are parsed and executed.

When initializing dynamic memory in the **MemoryPool** in the constructor do not forget to **set it's value to 0.**

If your implementation of the **MemoryPool** class is correct – the output to the standard output will be correct.
If your implementation is wrong – the output will be wrong.

The input reads integers numbers that represent commands listed in the **Defines.h** file

```
enum InputCommands
{
      REQUEST_MEMORY        = 0,
      RELEASE_MEMORY        = 1,
      ZERO_MEMORY            = 2,
      SUM_NODE_DATA         = 3,
      INCR_NODE_DATA_VALUE = 4
};
```

Keep in mind that the MemoryPool has a **Fixed** size of underlying memory.
Calls to **.requestMemory()** may succeed or fail depending on that if the MemoryPool has enough free memory.
The method returns an error code that is listed in the **Defines.h** file

```
enum ErrorCode
{
    REQUEST_SUCCESS                    = 0,
    REQUEST_FAILURE_NOT_ENOUGH_MEMORY  = 1,
    REQUEST_FAILURE_BIGGER_THAN_BUFFER = 2
};
```

Example: We have MemoryPool with size 5. `*****`

```
1) Request for memory with size 4 - ***** REQUEST_SUCCESS
2) Request for memory with size 2 - ***** REQUEST_FAILURE_NOT_ENOUGH_MEMORY
3) Request for memory with size 6 - ***** REQUEST_FAILURE_BIGGER_THAN_BUFFER
4) Request for memory with size 1 - ***** REQUEST_SUCCESS
5) Release memory for index 0 - ***** RELEASE_MEMORY for idx: 0
6) Request for memory with size 2 - ***** REQUEST_SUCCESS
7) Request for memory with size 1 - ***** REQUEST_SUCCESS
```

Important note: `RELEASE_MEMORY` should Not zero the memory.
Only calls to `ZERO_MEMORY` should zero the memory for the selected node.

Your task is to study the code and implement the function so that the code accomplishes the task described.

You should submit a single **`.zip`** file for this task, containing **ONLY** the files you created.

The Judge system has a copy of the other files and will compile them, along with your file, in the same directory.

Restrictions

You should only submit files with **.h** and **.cpp** extensions (in a **.zip** archive).

You should **Not** have folders in your **.zip** submission archive.

Examples

Input	Output

4 5
0 4
0 6
0 2
0 1
	REQUEST_MEMORY for size: 4, ErrorCode: REQUEST_SUCCESS
	REQUEST_MEMORY for size: 6, ErrorCode: REQUEST_FAILURE_BIGGER_THAN_BUFFER
	REQUEST_MEMORY for size: 2, ErrorCode: REQUEST_FAILURE_NOT_ENOUGH_MEMORY
	REQUEST_MEMORY for size: 1, ErrorCode: REQUEST_SUCCESS

6 5
0 5
3 0
4 0 2
3 0
2 0
3 0
	REQUEST_MEMORY for size: 5, ErrorCode: REQUEST_SUCCESS
	SUM_NODE_DATA for idx: 0, sum: 0
	INCR_NODE_DATA_VALUE for idx: 0, incrValue: 2
	SUM_NODE_DATA for idx: 0, sum: 10
	ZERO_MEMORY for idx: 0
	SUM_NODE_DATA for idx: 0, sum: 0

`8 7`
`0 3`
`0 4`
`4 1 3`
`4 0 2`
`1 0`
`1 0`
`0 5`
`3 0`
	`REQUEST_MEMORY for size: 3, ErrorCode: REQUEST_SUCCESS`
	`REQUEST_MEMORY for size: 4, ErrorCode: REQUEST_SUCCESS`
	`INCR_NODE_DATA_VALUE for idx: 1, incrValue: 3`
	`INCR_NODE_DATA_VALUE for idx: 0, incrValue: 2`
	`RELEASE_MEMORY for idx: 0`
	`RELEASE_MEMORY for idx: 0`
	`REQUEST_MEMORY for size: 5, ErrorCode: REQUEST_SUCCESS`
	`SUM_NODE_DATA for idx: 0, sum: 12`