C++ Programming: Exam Variant 2 (Exam-2017-05-28)

Solutions for each task will be submitted in the form of compressed archive (.zip) files, containing .h and .cpp files.

Please be mindful of the strict input and output requirements for each task, as well as any additional requirements on running time, used memory, etc., as the tasks are evaluated automatically and not following the requirements strictly may result in your program's output being evaluated as incorrect, even if the program's logic is mostly correct.

For some of the tasks in this exam you are provided with files, which the Judge system places in your submitted solution. These files are the so-called "Solution Skeleton" and, depending on the task, may require you to write specific code for your solution to work (e.g. a Solution Skeleton may contain a file with the **`main()`** function defined, in which case your task will usually be to implement a class or function in another file, for the program to work correctly). DO NOT attempt to edit the Solution Skeleton files – the Judge system overwrites any files from the skeleton you submit, so it won't see your changes to them. Some tasks may contain additional files you can use (and edit) if you want – if so, this will be described explicitly in the task.

You can use C++03 and C++11 features in your code.

Unless explicitly stated, any integer **input** fits into **`int`** and any floating-point **input** can be stored in **`double`**.

Task 4 – Calculator (E2-Task-4-Calculator)

You are given the skeleton of a calculator program (like the Calculator app in Windows, or the calculator on your smartphone, etc.). The program reads numbers and operations from the console and executes those operations on the numbers. The numbers are positive integers, while the operations can be single symbols (e.g. the star symbol '**`*`**' means multiplication), or strings of characters (e.g. the operation "**`end`**" stops the program and prints out the result).

Operations are executed immediately after they receive all their needed operands. For example, the expression **`3 * 4 / 2`** will first store **3**, then see the multiplication and wait for a number to multiply – when it receives **4** it will calculate **`3 * 4 = 12`**, then see the division and wait for a number to divide by – when it receives **2**, it will divide **12** by **2**.

Any number input overwrites the current result of the calculator, just like in normal calculators. For example, if the expression **`3 1 * 4 16 / 2`** is input, we'd first have **3**, overwrite it with **1**, multiply by **4** and get **4**, but then we overwrite with **16** and divide that by **2** – the result will be **8**.

The skeleton you are provided with contains the following files:

- **`main.cpp`** – contains the **`main()`** function, reads input and prints output on the console
- **`Operation.h`** – contains a base class for any operation done by the calculator
- **`MultiplicationOperation.h`** – defines a class which inherits the base **`Operation`** class and implements the multiplication operation (**`*`**)
- **`CalculationEngine`**.**`h`** – defines the calculator's central logic of handling number and operations input
- **`InputInterpreter.h`** – defines a class which can interpret a string into either a number or an operation and invoke the engine accordingly

You are also provided with the **`Extensions.h`** file, which is not part of the skeleton (you CAN edit it), but is used by **`main.cpp`** to initialize an **`InputInterpreter`**.

The files you are given support all logic necessary to implement the **multiplication** operation, as well as console input and output (note that input items don't need to be on the same line – you can write 1 operation or number per line and the code will still work).

Your task is to add the following operations:

- **`/`** – division, divides the current result of the calculator by the next number the calculator receives, and pushes the result to the calculator (i.e. same as multiplication, but divides)
- **`ms`** – saves the current result of the calculator to "memory". The result of this operation is the current result of the calculator. For example, the expression **`3 * 4 ms * 5`** and the expression **`3 * 4 * 5`** are equivalent in their result
- **`mr`** – memory recall, removes the last item from memory, and sends it to the calculator. Note that this operation can be used in combination with other operations, for example the expression **`3 ms * 4 ms * 5 * mr * mr`** will save **3** to memory, calculate to **12**, save to memory, calculate **60**, multiply that by **12** from memory, resulting in **720**, then multiply that by **3** from memory, resulting in **2160**. It can also be used without operations – **`3 ms 4 mr`** is the same as **`3 4 3`**

Input

The program defined in **`main.cpp`** reads the following input:

Strings, representing numbers or operations, separated by spaces (or new lines, or any "blank" space), ending with the string **end**.

Output

The program defined in **main.cpp** writes the following output:

The calculated result of all the numbers and operations from the input.

Restrictions

The numbers in the input will always be positive integers and no operation will result in a number larger than 1 billion.

There will always be at least 1 **ms** operation before any **mr** operation. There will be no more **mr** operations than the preceding **ms** operations. There will be no **ms** operation following an operation expecting a value (e.g. **3 * ms 4** is not a valid input, but **3 ms * 4** is). There will never be an invalid series of operations (e.g. **3 / / 4**, or **3 * * 4**, etc.)

The first **40%** of the tests will NOT contain **ms** or **mr** operations.

The total running time of your program should be no more than **0.1s**

The total memory allowed for use by your program is **5MB**

Submission Instructions

You CAN (and should) edit and submit the **Extensions.h** file. You should NOT submit any of the skeleton files – any implementation you do should be in the **Extensions.h** file, or other files you add.

Any file you submit should include all its dependencies (don't rely on what **main.cpp** has included)

Example I/O

Example Input	Expected Output
`1 * 2 * 3 ms * 4 * mr / 2 end`	72
`12 / 3 ms / 2 ms * 5 mr * mr end`	8