C++ Programming: Exam Variant 1 (Exam-2017-05-21)

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 – Word (E1-Task-4-Word)

You are given the skeleton of a word-processing program (like MS Word, OpenOffice Writer, etc.). The program reads a line of text from the console, then starts reading commands for editing (text-transform) and executing them on the text. Each command changes the text, the following command works on the changed text. When the command **exit** is entered, the program prints out the modified text and exits. All commands are of the form:  
**commandName startIndex endIndex**  
Where **commandName** is a string describing which command should be used, **startIndex** is an integer which describes from which index in the text the command should be applied, **endIndex** is an integer which describes to which index (exclusive) the command should be applied (i.e. the command is applied on indices starting from **startIndex** and ending in **endIndex - 1** inclusively)

The skeleton you are provided with contains the following files:

* **main.cpp** – contains the **main()** function, reads input and prints output on the console
* **TextTransform.h** – contains a base class for any text-transform added to the program
* **CommandInterface.h** – defines a base class which handles commands represented as strings (coming from the console, read from **main()**)

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

The files you are given support all logic necessary to implement the following command:

* **uppercase** – transforms any alphabetical character in the text in the range **[startIndex, endIndex)** to its uppercase variant.  
  E.g. if the current text is **som3. text**   
  and we are given the command **uppercase 1 7**  
  the current text will change to **sOM3. Text**  
  Note: if **startIndex == endIndex**, the command has no effect

Your task is to add the following commands:

* **cut** – cuts (removes) characters in the text in the range **[startIndex, endIndex)**, and remembers the last thing that was removed (Hint: **std::string::erase**)  
  E.g. if the current text is **som3. text**   
  and we execute the command **cut 1 7**  
  the current text will change to **sext** (… *I honestly didn’t plan in advance for this to be the result*)  
  Note: if **startIndex == endIndex**, the command has no effect on the text, but “clears” the last remembered cut
* **paste** – replaces the characters in the text in the range **[startIndex, endIndex)** with the characters which were removed by the last cut (Hint: **std::string::replace**)  
  E.g. if we have the text **som3. Text** and the commands  
  **cut 1 7** (text changed to **sext**)  
  **paste 3 4**  
  the current text will change to **sexom3. t**  
  (we paste the last cut – **"om3. t"** – over the **'t'** at the end of the text)  
  Note: if **startIndex == endIndex**, **paste** will insert the text at position **startIndex**, meaning that any text at **startIndex** will be pushed to the right by the inserted text. E.g. if the last command was **paste 0 0** (not **paste 3 4**), the text would be **om3. Tsext**

Input

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

A line of text, followed by a sequence of lines containing commands of the format   
**commandName startIndex endIndex**,   
ending with the command **exit**.

Output

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

The modified line of text.

Restrictions

The input text will be no more than **30** characters long and there will be no more than **10** commands in the input (this task is not about algorithm optimization).

For **currentTextLength** equal to the current number of characters in the text, for any command:  
**0 <= startIndex <= endIndex < currentTextLength**(i.e. the input will always be valid)

There will always be at least 1 **cut** command before any **paste** command. Consecutive **paste** commands (without **cut** between them) will paste the same text (just like in any text editor – you can cut something and paste it several times).

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 **Initialization.h** file. You should NOT submit any of the skeleton files – any implementation you do should be in the **Initialization.h** file, or other files you add.

Known Issues with the Judge System & Submission Advice

The Judge system has trouble compiling more than a few header files together. To be safe, submit only a single **Initialization.h** file with all your code in it (put any classes you implement there).

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 |
| som3. text  cut 1 7  paste 3 4  exit | sexom3. t |
| abc d e  cut 0 4  uppercase 1 3  paste 1 2  exit | dabc E |