Code Organization and Templates – Lab

This document defines the lab for the "C++ OOP" course @ Software University. Please submit your solutions (source code) to all below-described problems in Judge.

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

Code should compile under the C++03 or the C++11 standard.

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.

1. Compile

You are given some code, which reads a single string from the input and writes it to the output without any changes.

The code however doesn't compile. Figure out what file(s) needs to be created, and what file it needs to contain, so that the program compiles and runs as described.

You should submit a single **`.zip`** file for this task, containing ONLY the file(s) YOU created. The Judge system has a copy of the other files and will compile them, along with your file, in the same directory.

Examples

Input
example
Output
example

2. Censorship

Did you hear about article 13? You should.

You are given some code, which reads a line of strings (separated by single spaces) which describe "**copyrighted**" strings. Then it reads several lines (until a line containing the string "**end**" is entered), and checks each of them against the **copyrighted** words:

- If the line matches a copyrighted string, it is "**blocked**"
- Otherwise it is printed back on the output
- The last line "**end**" is not included in this logic, it just determines when the reading ends. No two lines will be the same.
- After the end line, the program prints a line beginning with "**`Blocked:`** ", followed by all the **blocked** strings, separated by single spaces, in the order they were entered

The program handles input and output, but uses a class named **`Article13Filter`** to do the blocking and store the blocked strings.

Your task is to study the available code and create a file with code such that the program compiles successfully and accomplishes the task described above.

You should submit a single **`.zip`** file for this task, containing ONLY the file(s) YOU created. The Judge system has a copy of the other files and will compile them, along with your file, in the same directory.

Examples

Input	Output
eussr censorship machines notblocked eussr machines hello censorship end	notblocked hello Blocked: eussr machines censorship

3. Parser

You are given code that reads objects from the console, each from a separate line, until a "**stopLine**" is entered. The format of the input is the following

- A character called **type**, which determines the type of each object – **`i`** for **integer**, **`w`** for **`string`**, and **`s`** for **Song**
- A line representing the **stopLine**
- Lines, each containing a single object, until a line matching the **stopLine** is reached – the program should not read anything after the **stopLine**.

The program then prints all the objects on a single line, separated by single spaces.

Your task is to study the available code and create the any necessary files for the program to compile and work correctly.

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 files, in the same directory.

Restrictions

Each type of object supports stream insertion **`operator<<`**

Each object is valid, meaning that each line representing an object can be turned into an object of the appropriate type through stream extraction **`operator>>`**

Examples

Input	Output
i 7 1 2 3 4 5 6 7 8	1 2 3 4 5 6
s ... caught-somewhere-in-time 446 superpalav 246 ...	caught-somewhere-in-time 446 superpalav 246

this is not printed	

4. Split and Join

You are given code, which reads the following

- A line containing a single character, called the **separator**
- Another **line** which needs to be split into **items** by the **separator**
- A character called **type**, which determines the type of each item – **`i`** for **integer**, **`w`** for **`string`**, and **`s`** for **Song**
- A string called a **joinStr**, which will be used to join the separated items together

The program handles input and output but uses a **`split`** and a **`join`** function to first split the **line** into **items** of the appropriate **type**, then join it back together into a single **`string`** and print it on the console. These need to be implemented for the program to compile and work correctly.

Your task is to study the available code and create the any necessary files for the program to compile and work correctly.

You should submit a single **`.zip`** file for this task, containing ONLY the file(s) 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

No item will contain the separator, i.e. the number of items is exactly equal to 1 + the number of separators.

Each item is valid, meaning that each item string can be turned into an item of the appropriate type through stream extraction **`operator>>`**

Examples

Input	Output
, 1,2,3,4,5,6 i ->	1->2->3->4->5->6
; caught-somewhere-in-time 446;superpalav 246 s >	caught-somewhere-in-time 446>superpalav 246

5. Sorting

NOTE: this task is the same as **Task 3 – Parser**, however the output is sorted depending on the object.

You are given code that reads objects from the console, each from a separate line, until a "**stopLine**" is entered. The format of the input is the following

- A character called **type**, which determines the type of each object – **`i`** for **integer**, **`w`** for **`string`**, and **`s`** for **Song**
- A line representing the **stopLine**
- Lines, each containing a single object, until a line matching the **stopLine** is reached – the program should not read anything after the **stopLine**.

The program then uses a **`set`** to store and then print all **unique** the objects on a single line, **sorted in descending order**, separated by single spaces. Descending order is different for each object type:

- For integers, it is the same as the standard C++ ordering of integers, in reverse, i.e. larger numbers are placed before smaller numbers
- For **`string`**s, it is the reverse of the C++ lexicographical order of **`string`**s
- For **Song**s, it is a reverse ordering by the **`lengthSeconds`** field, i.e. longer **Song**s are placed before shorter **Song**s

Your task is to study the available code and create any necessary files for the program to compile and work correctly.

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 files, in the same directory.

Restrictions

Each type of object supports stream insertion **`operator<<`**

Each object is valid, meaning that each line representing an object can be turned into an object of the appropriate type through stream extraction **`operator>>`**

Examples

Input	Output
i 7 1 2 3 4 5 6 7 8	6 5 4 3 2 1
s ... caught-somewhere-in-time 446 superpalav 246 ... this is not printed	caught-somewhere-in-time 446 superpalav 246