Objects and Classes Advanced – Exercise

Please submit your solutions (source code) to all below-described problems in Judge.

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

You are given a program in a **CarMain.cpp** file that reads and creates **const** objects of class **Car**, which has the following attributes:

- Brand (**string**)
- Model (**string**)
- Year (**int**)

The program reads input and writes output to the console, using the Car class and getters for the above-mentioned attributes.

Your task is to study the code in **CarMain.cpp** and implement the **Car** class in **Car.h** (which is **#include**-d by **CarMain.cpp**), so that **CarMain.cpp** compiles successfully and accomplishes the task described. Your **Car.h** file should resemble the following:

Car.h

```
#ifndef CAR_H
#define CAR_H

// Place your code here

#endif // !CAR_H
```

You should submit a single **.zip** file for this task, containing ONLY the **Car.h** file. 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
Volkswagen Golf 2015	Brand -> Volkswagen Model -> Golf Year -> 2015

2. Sum Of Vectors

You are given a program in a Main**.cpp** file that reads:

- How many couples of data do you have
- Elements data 1
- Elements data 2

The number of couples is an integer, the datas are strings.

With the couples in the data vectors, you have to create a new vector with the concatenation of the previous 2 vectors' data.

Write a function that does the concatenation

Your task is to study the code in **Main.cpp** and implement the **SumOfVectors** function in **SumOfVectors.h** (which is **#include**-d by **Main.cpp**), so that **Main.cpp** compiles successfully and accomplishes the task described. Your SumOfVectors**.h** file should resemble the following:

SumOfVectors.h

```
#ifndef SUMOFVECTORS_H
#define SUMOFVECTORS_H

// Place your code here

#endif // !SUMOFVECTORS_H
```

You should submit a single **.zip** file for this task, containing ONLY the **SumOfVectors.h** file. 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
2 Ivan Dragan 22 23	Ivan 22 Dragan 23
1 Polya Pavlova	Polya Pavlova

3. Operators

You are given code for a program that reads a specified number of lines from the console, and then formats them in the following way:

- It prepends two lines of characters before the lines from the input
- It then places each of the original lines prefixing it with its number (starting from 1) followed by **". "**
- It then appends a line of characters at the end

The program does all this (you don't have to do input, output, or determine line numbers), but it uses operators which aren't defined in the C++ language.

Your task is to study the code in **OperatorsMain.cpp** and implement the necessary operators in **Operators.h** (which is **#include**-d by **OperatorsMain.cpp**), so that **OperatorsMain.cpp** compiles successfully and accomplishes the task described. Your **Operators.h** file should resemble the following:

Operators.h

```
#ifndef OPERATORS_H
```

```
#define OPERATORS_H

#include <ostream>
#include <vector>
#include <string>
#include <sstream>

// Place your code here

#endif // !OPERATORS_H
```

You should submit a single **.zip** file for this task, containing ONLY the **Operators.h** file. 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
2 hello c++ operators	Formatted Lines ---------- 1. hello (5) 2. c++ operators (13) ----------

4. Resources

You are given code for a program that reads information about Resources (links to Presentations, Demos, or Videos) in the SoftUni Learning system and then sorts them by their id, and also prints how many of each type of resource there was in the input.

Each **Resource** has the following properties:

- An integer **id**
- A **ResourceType** – one of **Presentation**, **Demo**, or **Video**
- A **string** representing the link to the resource

The code for the program is in the **ResourcesMain.cpp** and it uses a **Resource** class, which it expects to be defined in a file named "**Resource.h**" in the same directory.

Your task is to study the **ResourcesMain.cpp** file and the **ResourceType.h** file, and to create the **Resource.h** file and implement the **Resource** class in such a way that the program correctly reads the input, orders it by id, prints it as lines on the output (each **Resource** output line should have the same format as the matching **Resource** input line) and then prints the number of Resources of each type.

Your **Resource.h** file should resemble the following:

Resource.h

```
#ifndef RESOURCE_H
#define RESOURCE_H

#include "ResourceType.h"

// Place your code here

#endif // !RESOURCE_H
```

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

Examples

Input
4 255 Demo http://kottakoa.com 42 Presentation http://theanswertolifetheuniverseandeverything.com 13 Demo http://example.com 69 Video http://yeahyouwish.com
Output
... by id: 13 Demo http://example.com 42 Presentation http://theanswertolifetheuniverseandeverything.com 69 Video http://yeahyouwish.com 255 Demo http://kottakoa.com ... by type: Presentation: 1 Demo: 2 Video: 1

5. Lectures

You are given code similar to **Task 4 – Resources**, however, this time the **main()** code uses a **Lecture** object (the class for which should be defined in a **Lecture.h** file) to store and organize the resources. It also uses several operators to do that and iterates the Resources in the Lecture through a range-based for loop (*hint: the **Lecture** class will need to **begin()** and **end()** methods which return iterators*).

Another difference is that in this task, there can be two **Resource** objects in the input which have different **links** but have the same **id**. This indicates that the Resource has been changed – i.e. if a **Resource** with the same **id** is encountered **multiple times**, only keep its version appearing **latest** in the input.

The program's output should be the same as in **Task 4** – the resources ordered by **id**, followed by the number of occurrences of each **ResourceType**, ordered **Presentation** (if non-zero), then **Demos** (if non-zero), and last **Video** (if non-zero).

Your task is to study the code, figure out what operators and classes you need to implement, search the Web for concepts you aren't familiar with, and submit the files necessary for the program to compile and run successfully.

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.

Examples

Input
6 42 Presentation http://thisiswillberenamed.com 255 Demo http://kottakoa.com 42 Presentation http://theanswertolifetheuniverseandeverything.com 13 Demo http://this.will.also.be.renamed.com 13 Demo http://example.com

69 Video http://yeahyouwish.com

Output
... by id: 13 Demo http://example.com 42 Presentation http://theanswertolifetheuniverseandeverything.com 69 Video http://yeahyouwish.com 255 Demo http://kottakoa.com ... by type: Presentation: 1 Demo: 2 Video: 1

Input
6 42 Demo http://thisiswillberenamed.com 255 Demo http://kottakoa.com 42 Demo http://theanswertolifetheuniverseandeverything.com 13 Demo http://this.will.also.be.renamed.com 13 Demo http://example.com 69 Video http://yeahyouwish.com

Output
... by id: 13 Demo http://example.com 42 Demo http://theanswertolifetheuniverseandeverything.com 69 Video http://yeahyouwish.com 255 Demo http://kottakoa.com ... by type: Demo: 3 Video: 1

6. Memory Allocator* (excluded from homework)

Your task is to write a simple memory allocator, which does not introduce a memory leak. You are given the **`main()`** function, which reads two values (as integer numbers) of memory followed by N command lines.

- The first integer value indicates the size of your memory allocator (in the range **`[0, INT_MAX]`** inclusive).
- The second integer value indicates the number of following command lines (N) you need to process and execute (in the range **`[0, INT_MAX]`** inclusive).
- The next **N** lines of indicating the command that you should process and execute.
  The commands have the following syntax:
  - **"Allocate INDEX"**
  - **"Deallocate INDEX"**
  - **"Idle"**

  Where INDEX can be any integer in the range (in the range **`[0, INT_MAX]`** inclusive);
  Keep in mind that INDEX may not be in present in your memory allocator boundaries.

You should implement the functions **executeCommand()** and **printResult()** in another .cpp file. (For example MemoryAllocator.cpp)

For each executed command in the **executeCommand()** – you should print a status message depending on the received **ErrorCode** in **printResult()**. Every call to **printResult()** should end with a **newline**.

You should print:

- For successful allocation/deallocation (not introducing memory leak or crashing the problem) – "**command** - success"
- For preventing a memory leak – "**command** - memory leak prevented, will not make allocation"
- For preventing a system crash – "**command** - system crash prevented, will skip this deallocation"
- For receiving an index that is not in the bound of your memory allocator – "**command** – out of bound"
- For receiving an "Idle" – "**command** - this exam is a piece of cake! Where is the OOP already?!?"
    - Where "**command**" is the exact same string that is passed to the function.

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 are free to implement another function/functions that are used internally by the **executeCommand()** and **printResult().**

Examples

Input	Output
5 2 Allocate 3 Deallocate 3	Allocate 3 - success Deallocate 3 - success
2 4 Deallocate 21 Allocate 1 Idle Allocate 1	Deallocate 21 - out of bound Allocate 1 - success Idle - this exam is a piece of cake! Where is the OOP already?!? Allocate 1 - memory leak prevented, will not make allocation
8 4 Allocate 2 Deallocate 2 Deallocate 2 Allocate 2	Allocate 2 - success Deallocate 2 - success Deallocate 2 - system crash prevented, will skip this deallocation Allocate 2 - success

7. Bytes Parsing* (excluded from homework)

Your task is to write program, which reads/represents/parses numbers out of contiguous array of bytes in memory into C++ primitive data type numbers. You are given the **`main()`** function, which reads two string values (as whole rows) of memory.

- The first string value indicates your command buffer. The buffer may only contain the letters ‘s’, ‘i’, ‘l’ (in any order and in any number of occurrences);
    - ‘s’ – stands for ‘short’ C++ primitive data type;
    - ‘i’ – stands for ‘integer’ C++ primitive data type;
    - ‘l’ – stands for ‘long long’ C++ primitive data type;
- The second string value contains your data buffer (as a single row of data).
  Each character of the second string will be in the range **[0, 9]** inclusive;
  After the read from the console an -= ‘0’ operation is performed on each char so the remaining value is the actual integer value from 0 to 9;
  Keep in mind that the command buffer may contain commands, which you have no data for in your data buffer. When you reach such a case -> simply ignore the rest of the commands from the command buffer.

You should implement the functions **parseData()** and **printResult()** in another .cpp file. (For example BytesParsing.cpp)

Keep in mind that the Judge system has a 64bit Little-endian architecture so:

- sizeof(short) is 2 bytes;
- sizeof(int) is 4 bytes;
- sizeof(long long) is 8 bytes;

Example:

command - "sil"

bufferData – "10200030000000"

- First parsed number is ‘short’ and first 2 bytes “[0-1]” are represented as an ‘short’;
- Second parsed number is ‘int’ and next 4 bytes from index [2-5] are represented as an ‘int’;
- Third parsed number is ‘long long’ and next 8 bytes from index [6-13] are represented as an ‘long long’;

The result is “1 2 3”

Example 2:

command - "silll"

bufferData – "10200030000"

The result is "**`1 2 Warning, buffer underflow detected`**"

The data buffer does not have enough information about all the listed commands. All the parsed numbers so far are printed first.

As a result of **parseData()** – you should print a status message depending on the received **ErrorCode** in **printResult()** followed by a **newline**.

You should print:

- For successful allocation parsing – all parsed numbers are divided by **whitespace** (the last number should also have whitespace before the newline);
- For partial parsing (more requested commands than actual data to parse) – all **successfully** parsed numbers so far divided by **whitespace** followed by "Warning, buffer underflow detected" (the last number should also have whitespace before the "Warning part");
- For preventing an empty command buffer or data buffer – "No input provided";

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 are free to implement another function/functions that are used internally by the **parseData()** and **printResult().**

The command buffer and data buffer size will be in the range **[0, SIZE_T_MAX]** inclusive;

Note: 'size_t' and 'unsigned long long integer' are the same thing;

Examples

Input	Output
ss 2002	2 512
is 11110	16843009 Warning, buffer underflow detected
sil 900200003000000	9 512 196608