# Exercise: OOP Constructors

Please submit your solutions (source code) to all below-described problems in [Judge](https://alpha.judge.softuni.org/contests/oop-constructors-exercise/3078).

**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.**

## TryParse

You are given a program in a Main.cpp file that reads two strings, each of which **is either a valid integer** **or** **contains only letters**, then attempts to parse them into **integer** numbers – using a function called tryParse– and calculate their sum.

If both numbers can be parsed, their sum is printed.

Otherwise, two lines are printed, one of them for the first number, the other for the second number. Each of those lines contains either the input string for that number, but if the number was not successfully parsed, the line starts with "[error] " followed by the input string for the number.

Your task is to study the code in Main.cpp, then create a file called TryParse.h (which Main.cpp includes) containing the definition of the tryParse function, written in such a way that Main.cpp compiles successfully and works as described above.

You should submit a single .zip file for this task, containing ONLY the TryParse.h file. The Judge system has a copy of the Main.cpp file and will compile it and your TryParse.h file in the same directory.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 13 42 | 55 |
| asd 112 | [error] asd  112 |

## Find

You are given a program in a Main.cpp file that reads info about **companies** – **name** and **id** – and then reads a **search id**, finds a company with that **id**, and prints the info about that company. If no such company has been entered, the program prints "[not found]". To do the search, the program uses a function named to find. To describe a company, it uses the Company.h file, which you are also given.

Your task is to study the code in Main.cpp, then create a file called Find.h (which Main.cpp includes) containing the definition of the find function, written in such a way that Main.cpp compiles successfully and works as described above.

You should submit a single .zip file for this task, containing ONLY the Find.h file. The Judge system has a copy of the Main.cpp file and will compile it along with your Find.h file in the same directory.

To correctly use the Company definition, without interfering with its usage by Main.cpp, use the following structure for the Find.h file:

|  |
| --- |
| **Find.h** |
| #ifndef FIND\_H  #define FIND\_H  #include "Company.h"  // Place your code here, as well as any other #include directives you might need  #endif // !FIND\_H |

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| acme 424242420  itjoro 878968302  softuni\_foundation 20140414  end  878968302 | itjoro 878968302 |

## Order

You are given a program in an OrderMain.cpp file that reads info about **companies** – **name** and **id** – and then prints them ordered by their id, in increasing order. To describe a company, it uses the Company.h file, which you are also given.

To order the companies, OrderMain.cpp uses a class named OrderedInserter from a file named OrderedInserter.h – it initializes it with an empty vector of companies, then calls a method named insert for each company in the input, then prints the contents of the vector.

Your task is to study the code in OrderMain.cpp, then create a file called **OrderedInserter**.h (which OrderMain.cpp includes) containing the definition of the **OrderedInserter** class, written in such a way that Main.cpp compiles successfully and works as described above.

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

To correctly use the Company definition, without interfering with its usage by OrderMain.cpp, use the following structure for the **OrderedInserter**.h file:

|  |
| --- |
| **OrderedInserter.h** |
| #ifndef ORDERED\_INSERTER\_H  #define ORDERED\_INSERTER\_H  #include "Company.h"  // Place your code here, as well as any other #include directives you might need  #endif // !ORDERED\_INSERTER\_H |

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| acme 424242420  softuni\_foundation 20140414  itjoro 878968302  end | softuni\_foundation 20140414  acme 424242420  itjoro 878968302 |

## Profits

You are given a program in an ProfitsMain.cpp, as well as a Company.h and ProfitCalculator.h file, that read info about **companies** – **name**, **id**, **revenue** and **costs**, followed by info about profit calculations per company – **company id** followed by a **tax percentage** – and generates a report with the profit for each company in the input.

The report must contain exactly as many lines as there are companies, and each line should contain the name of the company on that line in the input, followed by a space, a '=', another space, and an integer value representing the profits of the company, e.g. a line of the output for a company called TheCompany with a profit of 42000 should look like this:

TheCompany = 42000

To generate the report, ProfitsMain.cpp uses a function named getProfitReportfrom a file named ProfitReport.h. The getProfitReport function receives 3 parameters:

* A pointer to the first company in an array of companies
* A pointer to the last company (inclusive) in an array of companies
* An std::map, which maps company ids to ProfitCalculators

The getProfitReport should use the appropriate ProfitCalculator from the map (i.e. the ProfitCalculator in the entry with a key matching the id of the company) to calculate each company’s profit.

The getProfitReport returns a string, containing the report for the provided companies, calculated through the provided ProfitCalculators, as described above.

Your task is to study the code in ProfitsMain.cpp, then create a file called **ProfitReport**.h (which ProfitsMain.cpp includes) containing the definition of the **getProfitReport** function, written in such a way that ProfitsMain.cpp compiles successfully and works as described above.

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

To correctly use the Company definition, and the ProfitCalculator definition, without interfering with their usage by ProfitsMain.cpp, use the following structure for the **OrderedInserter**.h file:

|  |
| --- |
| **ProfitReport.h** |
| #ifndef PROFIT\_REPORT\_H  #define PROFIT\_REPORT\_H  #include "Company.h"  #include "ProfitCalculator.h"  // Place your code here, as well as any other #include directives you might need  #endif // !PROFIT\_REPORT\_H |

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| acme 424242420 : 43000 1000  softuni\_foundation 20140414 : 0 0  itjoro 878968302 : 100 25  end  878968302 0  424242420 10  20140414 30  end | acme = 37800  softuni\_foundation = 0  itjoro = 75 |

## Register

You are given code that reads information about Company objects from the console, parses it multiple times (the number of repetitions is entered on the first line on the console), and prints the information about one of the Company objects, specified by its **id**.

The provided code handles input, output, and the repeated executions – your task is to **implement** the Register (which is declared in the Register.h file, you need to create the Register.cpp file) class it uses for storing and looking up the Company objects.

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

There will always be a Company with the specified **id**.

Make sure there are no memory leaks.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1  2  42 theanswer  69 thehub  42  end | 42 theanswer |
| 1000  2  42 theanswer  69 thehub  42  end | 42 theanswer |

## Words

You are given code that reads two lines of words (**strings** containing lowercase English letters, separated by spaces) and prints the number of occurrences of each word (in lexicographical order, as C++ orders **string**s) in the first input line, then does the same for the second input line.

The provided code handles input and output, however, it uses a Word class for the counting. Your task is to implement the Word class so that the program compiles successfully and runs as described.

NOTE: the **main()** function just reads and initializes objects of the **Word** class, then adds them to a set to sort them lexicographically. It does not call any methods, other than the ones for getting the word string and the count for it at the end. You need to figure out how to handle the counting based on the provided code.

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** |
| she sells sea shells on the sea shore  hello world | on 1  sea 2  sells 1  she 1  shells 1  shore 1  the 1  ---  hello 1  world 1 |
| she sells sea shells on the sea shore  the shells she sells are very shiny | on 1  sea 2  sells 1  she 1  shells 1  shore 1  the 1  ---  are 1  sells 1  she 1  shells 1  shiny 1  the 1  very 1 |

## Divisible by 45

You are given a BigInt.h file with the implementation of a BigInt class which can represent positive integers of any size, can calculate sums of such integers, and has some other useful methods and operators defined. You can use this class in your program if you want to.

* The Judge system has a copy of this class and will compile your code in the same directory.
* To use it, you can write #include "BigInt.h" in your code.
* DO NOT submit or modify BitInt.h, as the system will overwrite it with its version. If you want to extend the functionality of that class, you will need to do it in another file.

Your task is to write a program that finds all the numbers, which are divisible by 45, inside a specified range.

For this task the system only has a copy of the BigInt.h file, so the .zip file you upload should contain a file with the main() function, and you should handle input and output as described below.

### Input

Exactly 2 lines, each containing a single integer number – with an arbitrary length, but no more than 100 digits.

The first line contains the start of the range (inclusive) S

The second line contains the end of the range (exclusive) E

### Output

One or more lines, with a single integer number each, representing the numbers divisible by 45 in the given range, in ascending order (i.e. start from the smallest number divisible by 45 in the range and print each of them on a separate line).

### Restrictions

The range will be such that the total numbers divisible by 45 will be no more than 100.

The number of digits in the numbers specifying the range will NOT exceed 100.

0 < S < E - 1

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.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1  100 | 45  90 |
| 1  90 | 45 |
| 450000000000000000000000000000000000000013  450000000000000000000000000000000000000100 | 450000000000000000000000000000000000000045  450000000000000000000000000000000000000090 |

## Sequences

You are given code for a program that uses a Sequence class that can be iterated with a range-based for loop and can generate its elements using a templated Generator type, which overloads parameter-less operator() to generate the next element in the sequence. The provided code has two Generator types – IntegersGenerator and FibonacciGenerator, respectively generating the integer numbers starting from 0 and the Fibonacci numbers starting from 0.

The provided code reads a number from the console, generates that amount of elements in the sequence, prints the first element, then reads a number again, generates that amount of new elements in the sequence, prints the second element, and so on, and continues until no more generated elements remain (meaning that the input always ends with 0).

Your task is to implement the Sequence class to support the described operations 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(s), in the same directory.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| i  3 1 1 0 2 0 0 0 | 0 1 2 3 4 5 6 |
| f  1 5 0 0 2 0 0 0 0 | 0 1 1 2 3 5 8 13 |