# 07. Destructors, Constructors and Copy-Assignment

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

Submit your solutions here: <https://judge.softuni.bg/Contests/1250/07-Destructors-Constructors-and-Copy-Assignment>

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.

# Task 1 – Register

You are given code which 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 |

# Task 2 – Register of Three

**NOTE**: this task is the same as **Task 1 – Register**, however the main() function in the skeleton is different and requires you to implement the Rule of Three for the Register class.

You are given code which 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 |

**Task 3 – Array of Pointers**

**NOTE**: this task is the same as **Task 1 – Register**, and **Task 2 – Register of Three,** however it does NOT use a Registerclass, but instead allocates **Company** objects in dynamic memory and uses an **ArrayOfPointers** class to store these pointers (NOTE: the provided code does NOT deallocate the memory it allocates).

You are given code which 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 **declare** and **implement** the **ArrayOfPointers** class in any way you think will accomplish the task, **without leaking memory**. The provided code expects the declaration to be in an **ArrayOfPointers.h** file, but you are free to chose whether to implement the class in the same file, in a **.cpp** file, or in multiple files.

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 |

**Task 4 – 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 |

**Task 5 – List**

You are given a **List.h** file containing the declarations for a **List** class representing a linked list, and a **ListMain.cpp** file, which defines a **main()** function and uses the **List** class to merge several sorted lists from the standard input into a single sorted list printed on the standard output.

* Create a **List.cpp** file which contains the implementation of the **List** class
* The files should successfully compile together
* The resulting program should correctly merge sorted lists read from the console into a sorted list, which should be printed on the console (main.cpp does this if you implement List.cpp)
* Submit a **.zip** file containing the **List.cpp** file and nothing else

How you choose to implement the linked list is up to you, but you should make sure all the public methods of the **List** class work correctly, as they are used by main.cpp. The declarations in **List.h** should be mostly self-explanatory, but if you are unsure what a method should do – just see how **ListMain.cpp** uses it and make sure you implement it so that the program works correctly.

You are NOT allowed to modify **ListMain.cpp** or **List.h**.

The task this program solves is merging multiple sorted (ascending) lists of integer numbers into a single sorted (ascending) list of integer numbers. For example, the lists **1 17**, and **-3 6 25 42** should be merged into the following list: **-3 1 6 17 25 42**.

**Input**

One or more lines, each of which containing from **1** to **100** integers, separated by single spaces. The final line will not contain numbers and will only contain the string **"end"**

**Output**

A line containing the items of the merged, sorted list, in ascending order, separated by single spaces.

**Restrictions**

The total number of elements entered in the input will NOT exceed **10000**

The number of elements per input list (line) will NOT exceed **100**

Numbers in the input will be from **-9999** to **+9999** (both inclusive)

The total running time of your program should be no more than **0.5s**

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

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 17  -3 6 25 42  end | -3 1 6 17 25 42 |
| 4  5 6  1 2 3  end | 1 2 3 4 5 6 |
| 1 3  2  end | 1 2 3 |