# 01. Pointers and References – Homework Exercises

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

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

Submit your solutions here: <https://judge.softuni.bg/Contests/1150/01-Pointers-and-References-Homework>

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 5 – Memory

You are given program in an MemoryMain.cpp, as well as a Company.h file, that read information about **companies** and writes it to the console.

Each company has:

* An **id** (an integer between 0 and 255)
* A **name** (a string containing a sequence of lowercase English letters a-z)
* **Employees** by their initials (a vector of pairs of characters, containing at most 255 employee initials)

The MemoryMain.cpp file reads the information from the console, as a sequence of byte values, stores those bytes in memory (RAM), and then calls a function named readCompaniesFromMemory, passing it two parameters:

* a **pointer** to the **first byte** in the memory containing the companies
* an integer indicating the **number of companies** stored in the memory

The memory format of each company is the following:

* the first byte contains the **id** of the company (0-255)
* the **name** of the company starts from the second byte and ends with a null terminator (the value 0, or '\0'), i.e. the name of the company is placed in memory the same way a null-terminated C-String would be
* the next byte contains the number of employees the company has (0-255). Let’s call it numEmployees
* the following numEmployees \* 2 bytes contain pairs of initials of the employees, i.e. if the numEmployees byte is at address x, then the **first employee’s first initial** is at address x + 1, their **second initial** is at address x + 2, the **second employee’s first initial** is at address x + 3 and their **second** is at address x + 4 and so on.

The MemoryMain.cpp file will print the companies in the format:

* company **id**, space, company **name**, space, opening bracket '(', first initial of first employee, dot **'.'**, second initial of first employee, dot '.', first initial of second employee, … , closing bracket ')'

For example, if we have the companies:

* id = 42, name = "uni", employees = { {'I', 'K'}, {'S', 'N'} } and   
  id = 13, name = "joro", employees = { {'G', 'G' } }

Their representation as strings printed by MemoryMain.cpp will be:

42 uni (I.K.,S.N.)  
13 joro (G.G.)

Their representation in memory, assuming the memory starts at byte address M**,** will be:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Offset from start** | **0** | **+1** | **+2** | **+3** | **+4** | **+5** | **+6** | **+7** | **+8** | **+9** | **+10** | **+11** | **+12** | **+13** |
| **Value** | **42** | **'u'** | **'n'** | **'i'** | **'\0'** | **2** | **'I'** | **'K'** | **'S'** | **'N'** | **13** | **'j'** | **'o'** | **'r'** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Offset from start** | **+14** | **+15** | **+16** | **+17** | **+18** |
| **Value** | **'o'** | **'\0'** | **1** | **'G'** | **'G'** |

And their representation in the input for the task will be:

42 117 110 105 0 2 73 75 83 78

13 106 111 114 111 0 1 71 71

**end**

Your task is to create a file called CompanyMemoryUtils.h (which MemoryMain.cpp includes), containing the function readCompaniesFromMemory, implemented in such a way that MemoryMain.cpp compiles successfully and works as described above – i.e. your task is to read the memory, which will be in the format described above, and return a vector<Company> containing the companies that were written in that memory.

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

NOTE: you are also given the code for the test generator used to generate the tests in the judge system, in C#. Compiling and running it will produce random tests (.in.txt input files and .out.txt output files) similar to those in the Judge system, which you can use to test your code locally.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 42 117 110 105 0 2 73 75 83 78  13 106 111 114 111 0 1 71 71  end | 42 uni (I.K.,S.N.)  13 joro (G.G.) |
| 188 105 99 121 104 97 0 3 66 81 72 80 70 83  58 117 97 100 101 108 0 3 83 65 67 72 76 84  end | 188 icyha (B.Q.,H.P.,F.S.)  58 uadel (S.A.,C.H.,L.T.) |