# 01. Pointers and References – Homework Exercises

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

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 1 – TryParse

You are given 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 |

# Task 2 – Find

You are given 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 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 |

# Task 3 – Order

You are given 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 |

# Task 4 – Profits

You are given 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:

### 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 |

# 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.) |