# 02. Memory Management – Homework Exercises

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

Submit your solutions here: <https://judge.softuni.bg/Contests/1152/02-Memory-Management-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 – MinBy

You are given code for a program, which finds the minimum element of a list of strings entered on the console (lowercase English letters, separated by spaces). The program has 3 modes of operation, each with using a different characteristic for finding the minimum:

* Lexicographical minimum i.e. finds the string that is earliest lexicographically
* Size minimum i.e. finds the shortest string
* Reverse size minimum i.e. finds the longest string

The code is missing the MinBy.h file. Your task is to study the provided code and create a MinBy.h file such that the program compiles successfully and performs the described task.

Your MinBy.h file should resemble the following:

|  |
| --- |
| **MinBy.h** |
| #ifndef MIN\_BY\_H  #define MIN\_BY\_H  // Place your code here  #endif // !MIN\_BY\_H |

You should submit a single .zip file for this task, containing ONLY the MinBy.h file. 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 at least1 string in the list of strings entered from the console. There will be no empty strings.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| hear the rime of the ancient mariner see his eyes as he stops one of  3 | ancient |
| e abc df e  1 | abc |

# Task 2 – Remove Invalid

You are given code for a program, which reads Company objects from the console, allocates dynamic memory for them, adds them to a list, and calls a removeInvalid function multiple times during the read and once after all companies have been read, to **remove companies with invalid ids**. The program then **prints all companies with a valid id**. Valid ids are **non-negative**.

Your task is to implement the removeInvalid function a RemoveInvalid.h file. The function should remove all companies with negative ids (use the getId() getter in Company) from the list.

Your RemoveInvalid.h file should resemble the following:

|  |
| --- |
| **RemoveInvalid.h** |
| #ifndef REMOVE\_INVALID\_H  #define REMOVE\_INVALID\_H  #include "Company.h"  // Place your code here  #endif // !REMOVE\_INVALID\_H |

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

### Additional Requirements

The order of the valid companies in the input must match the order in the output.

Make sure there are no memory leaks.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 123 valid  -123 invalid  321 ancientmarinerinc  end | 123 valid  321 ancientmarinerinc |
| 42 noinvalidhere  end | 42 noinvalidhere |

# Task 3 – Make Company

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

Each company has:

An **id** (a 4-byte integer)

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 MakeCompanyMain.cpp file reads the information from the console as a single line for each company, containing:

* company **id** string
* space
* company **name**
* 2 characters representing the initials of the first employee
* Space
* Again 2 characters, representing the second employee, etc

NOTE: there could be companies without employees, in which case the line ends with the company name.

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 read by MemoryMain.cpp will be:

42 uni IK SN  
13 joro GG

The program reads each line from the console, then calls a function named makeCompany to **dynamically allocate memory** for a Company object and set its values, then prints its representation as a back to the console (i.e. the program should print the exact line it read), until a line containing the single string "end" is reached.

Your task is to create a file called MakeCompany.h containing the function makeCompany such that the program compiles successfully and performs the described task.

Your file should resemble the following:

|  |
| --- |
| **MakeCompany.h** |
| #ifndef MAKE\_COMPANY\_H  #define MAKE\_COMPANY\_H  #include "Company.h"  // Place your code here  #endif // !MAKE\_COMPANY\_H |

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

### Additional Requirements

Make sure you correctly create the Company objects and ensure that there are **no memory leaks**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 42 uni IK SN  13 joro GG  end | 42 uni IK SN  13 joro GG |
| 188 noemployees  58 oneemployee SA  end | 188 noemployees  58 oneemployee SA |

# Task 4 – Parse Unique Companies

You are given code for a program, which reads lines from the console containing the information about Company objects, sends those lines as a single string to a function named parseUniqueCompanies and expects the function to return a pointer to a dynamically allocated array, which the program iterates and prints information about each parsed Company.

In the input, each Company is described by a single line, containing the Company id (an integer) followed by the Company name (a string), separated by a single space.

In addition to the string, the program passes two other parameters:

* an integer, which it expects to be set (by the parseUniqueCompanies function) with the **number of companies** in the input
* a function pointer/reference to a function that generates a unique identifier for a company. The parseUniqueCompanies function is expected to return no more than one company (the first in the input) for each unique identifier

The program decides which unique identifier function to use based on a number in the last line of the input:

* unique identifier is the Company’s id
* unique identifier is the Company’s name
* unique identifier is the Company’s name concatenated by the Company’s id

Your task is to implement the parseUniqueCompanies function a ParseCompanies.h file.

Your ParseCompanies.h file should resemble the following:

|  |
| --- |
| **ParseCompanies.h** |
| #ifndef PARSE\_COMPANIES\_H  #define PARSE\_COMPANIES\_H  #include "Company.h"  // Place your code here  #endif // !PARSE\_COMPANIES\_H |

You should submit a single .zip file for this task, containing ONLY the ParseCompanies.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** |
| 321 valid  123 copy  123 ancientmarinerinc  end  1 | 123 copy  321 valid |
| 42 theanswertolifetheuniverseandeverything  420 theanswertolifetheuniverseandeverything  end  3 | 42 theanswertolifetheuniverseandeverything  420 theanswertolifetheuniverseandeverything |

# Task 5 – Serialize

NOTE: this task is the reverse of Task 5 – Memory from the lecture 01. Pointers and References.

You are given program that reads 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 program reads the information in its string representation and calls a function named serializeToMemory. The function should parse the companies from the input and then write them to memory as a sequence of bytes in a dynamically allocated array (the format is detailed below). The function will be called with the following two parameters:

* A string containing lines, where each line is the string representation of a Company
* An integer which the program expects to be set with the number of bytes serialized to memory which contain the representation of the companies from the first parameter

The program expects the function **serializeToMemory** to return a pointer to the memory where the companies have been written (serialized).

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.

Additionally, since there can be more than one Company:

* the **first byte** in the memory describing the companies contains an **integer representing the number of companies** serialized

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 a string read by the program and passed to **serializeToMemory** will be:

"42 uni (I.K.,S.N.)\n13 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** | **+14** |
| **Value** | **2** | **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 output for the task will be:

2 42 117 110 105 0 2 73 75 83 78 13 106 111 114 111 0 1 71 71

Your task is to create a file called Serialize.h containing the function **serializeToMemory**, implemented in such a way that the program compiles and works as described.

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

Hint: the Company class supports reading from a stream, so you don’t need to implement the parsing of the string yourself. The following code reads companies from the **string**, until there are no more companies to read:

std::istringstream companiesIn(companiesString);

Company company;

while (companiesIn >> company) { }

### Examples

|  |  |
| --- | --- |
| **Input** | **Output (*NOTE: single line*)** |
| 42 uni (I.K.,S.N.)  13 joro (G.G.)  end | 2 42 117 110 105 0 2 73 75 83 78 13 106 111 114 111 0 1 71 71 |
| 188 icyha (B.Q.,H.P.,F.S.)  58 uadel (S.A.,C.H.,L.T.)  end | 2 188 105 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 |
| 13 joro (G.G.)  end | 1 13 106 111 114 111 0 1 71 71 |