ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecturer: Prof. B. Gärtner

Informatik für Mathematiker und Physiker HS15

Exercise Sheet 12

Submission deadline: 15:15 - Tuesday 8th December, 2015

`Course URL`: http://lec.inf.ethz.ch/ifmp/2015/

Assignment 1 (4 points)

Explain each of the following concepts in your own words. Your explanations shall include the concepts provided in brackets. A possible answer to subtask e) is already provided as an example.

a) `class` (`struct`, `public`, `private`)

b) `constructor` (`overload`, `default-constructor`, `conversion`)

c) `copy-constructor` (`operator=`, `dynamic data type`)

d) `destructor` (`dynamic data type`)

e) `new` (`delete`, `pointer`, `dynamic data type`)

For example e): `new` is used to dynamically allocate memory (i.e. the amount is only known when the program is running). `dynamic data types` - which can dynamically expand and shrink - depend on this dynamic memory allocation. `new` exists in a version for single elements (`new int (3)` for example) and a version for ranges (`new int[n]` for example). Both have in common that they return `pointers` to the allocated space (the first element of it in the case of ranges). When the space is no longer required it must be explicitly deleted using `delete` for single elements and `delete[]` for ranges.

Assignment 2 (4 points)

In this exercise we will implement operations for 2×2 matrices with elements of type `double`. Again try to reuse as much code as possible.

a) Implement a class `Matrix` to represent 2×2 matrices with entries of type `double`.

b) Implement the following matrix operations for your class: matrix addition, matrix multiplication, and scalar multiplication (scalar of type `double`) for matrices.

c) Implement `operator>>` and `operator<<` for your `Matrix` type. The following format shall be used:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \text{as} \quad a\ b\ c\ d$$

d) Implement a member function `det` to compute the determinant of matrices.

e) Implement a `main` function for your program which reads two matrices `A` and `B`, and a factor `c` of type `double` from the user. Your program should then output `A+B`, `A*B`, `c*A`, `B*c`, and `A.det()`.

Judge Examples (Explanation: http://lec.inf.ethz.ch/ifmp/2015/judge_boxes.html)

```
Input first matrix: 0 0 0 0
Input second matrix: 1 2 3 4
Input scalar c: 48
A*B = 0 0 0 0
A+B = 1 2 3 4
c*A = 0 0 0 0
B*c = 48 96 144 192
A.det() = 0
```

```
Input first matrix: 1.5 2.5 3.25 4.25
Input second matrix: 8 3.5 2 7.25
Input scalar c: 0.5
A*B = 17 23.375 34.5 42.1875
A+B = 9.5 6 5.25 11.5
c*A = 0.75 1.25 1.625 2.125
B*c = 4 1.75 1 3.625
A.det() = -1.75
```

Submission: https://challenge.inf.ethz.ch/team/websubmit.php?cid=5&problem=MP15122

Assignment 3 (4 points)

The deadline of this exercise is expanded by another week.

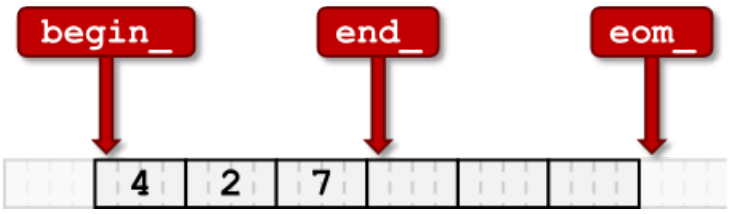
In this exercise we ask you to implement a class `ifmp::vector` for type `int` that provides *similar* functionality as `std::vector<int>`, at least as far as dynamic memory management is concerned. Internally, the class `std::vector` uses *dynamically* allocated arrays to store the elements. Like for normal arrays, the elements are stored in contiguous locations in the memory. This allows for efficient access of the elements with pointer arithmetic.

The class `std::vector` can not only allocate memory of arbitrary size at the moment it is created, but it can also dynamically change its size. For instance, you can add an element `e` to the end of the vector with the function `push_back(e)`, and the size increases by 1. To add an element at the end of a vector of size s, it is not enough to simply reserve some memory for this single element and store it there. The memory occupied by the vector might not be contiguous. Instead you will have to reserve a larger range in memory and copy over the entire vector to the new range. It would be a wise decision at this point to reserve a range which is larger by more than one element (good choice is: larger by a factor of 2, i.e. $2s$ elements) and hide the additional elements from the user. This way, later `push_back`

calls can simply write into these spare elements instead of having to copy over the entire vector to a larger range each time `push_back` is called.

Download the file `vector_template.cpp` from the website and implement the member functions of `ifmp::vector`. Of course you are not allowed to use any data structure from the Standard Library that already provides dynamic memory allocation, but you should call `new` and `delete` directly.

Note: The class `ifmp::vector` stores three pointers: `[begin_, end_of_memory_)` denotes the range of *allocated* memory, and `(end_of_memory_-begin_)` is called the *capacity* of the vector. `[begin_, end_)` denotes the range of *used* memory by the elements of the vector, and `(end_-begin_)` is called the *size* of the vector. The following illustration depicts the pointer layout. (The name `end_of_memory_` is shortened to `eom_`.)



Judge Examples (Explanation: http://lec.inf.ethz.ch/ifmp/2015/judge_boxes.html)

```
Input numbers: -8 3 2 0
Size = 4
size <= capacity ? 0

Output []: -8 3 2 0
Output itr: -8 3 2 0
Other Output: 2 2 2 -8 3 2 0
```

```
Input numbers:
Size = 0
size <= capacity ? 0

Output []:
Output itr:
Other Output: 2 2 2
```

Submission: https://challenge.inf.ethz.ch/team/websubmit.php?cid=5&problem=MP15123

Challenge – Skript-Aufgabe 149 (8 points)

Do you dream of creating virtual reality? This challenge is the easiest way to get you started! You will visualize and rotate a three-dimensional object, using the graphics library built into the VirtualBox. And since it's virtual, you are free to add an endless list of cool features (growing, shrinking, squeezing,...)!