ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecturer: Prof. B. Gärtner

Informatik für Mathematiker und Physiker HS14

Exercise Sheet 12

Submission deadline: 15:15 - Tuesday 9th December, 2014
`Course URL:` http://lec.inf.ethz.ch/ifmp/2014/

Assignment 1 – Skript-Aufgabe (4 points)

In three-valued logic the type `bool` is extended with a third state, namely `unknown`. And as for `bool` we can also define logical operators for three-valued logic. For example:

AND	false	unknown	true
false	false	false	false
unknown	false	unknown	unknown
true	false	unknown	true

OR	false	unknown	true
false	false	unknown	true
unknown	unknown	unknown	true
true	true	true	true

Write a class `Tribool` which models three-valued logic, and which has

a) only private *data* members,

b) an overload of `operator<<`,

c) `operator&&` and `operator||` (they shall also work if one operand is of type `bool`),

d) an access function `is_bool() const` that returns *true* if and only if the value is not *unknown*,

e) conversions from the types `bool` and `int` to `Tribool`.
For bool: `true --> true, false --> false`
For int: `1 --> true, 0 --> false, else --> unknown`.

There is a template (`tribool_class_template.cpp`) available on the lecture website which you can use to solve this exercise if you want.

Assignment 2 (4 points)

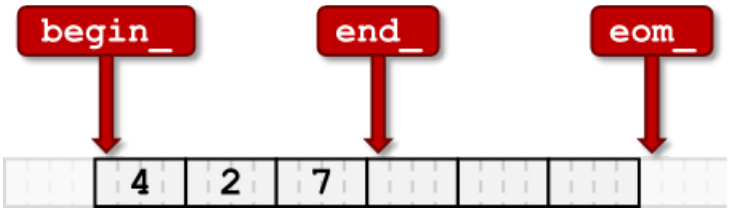
In this exercise we ask you to implement a class `ifmp::vector` for type `int` that provides *similar* functionality as `std::vector<int>`, at least as far as dynamic memory management is concerned. Internally, the class `std::vector` uses *dynamically* allocated arrays to store the elements. Like for 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 files `vector.h`, `vector.cpp`, and `vector_test.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_`.)



Assignment 3 (4 points)

Reverse polish notation (RPN) is a way to write arithmetic expressions without brackets. As an example, consider

$$4\ 2 - 5\ 4 + *.$$

RPN expressions are evaluated from left to right. When the next item is an operand n (a number), n is pushed onto a stack that is initially empty; when the next item is an operator op $(+, -, *, /)$, the top element n_2 and the one below it, n_1, are popped from the stack, and the result $n_1\ \mathrm{op}\ n_2$ is pushed back to the stack. In case of a valid RPN expression, the stack contains exactly one number in the end, and this is the value of the expression. In the example above, the evaluation yields $(4-2)*(5+4) = 18$.

Write a program that evaluates expressions in RPN over operands of type `double`! An expression should be read from an input stream until the stream becomes empty. Use assertions to "catch" illegal expressions (less than two operands on the stack for some operator, or not exactly one operand on the stack in the end). You may either use the type `std::stack<double>` for your program, or adapt the type `ifmp::stack` from the lecture for this task.

On the course page, you find a template `rpn_template.cpp` that you can use to get started.

Challenge – Skript-Aufgabe 157 (8 points)