ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecturer: Prof. B. Gärtner

Informatik für Mathematiker und Physiker HS15

Exercise Sheet 13

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

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

Assignment 1 – Skript-Aufgabe 159 (4 points)

Implement an extended variant of the class ifmp:stack that in addition to the public member functions push, pop, top, and empty also has the following public member:

```
// POST: number of elements in *this is returned
unsigned int size () const;
```

In addition, implement an equality test for stacks

```
// POST: returns true if and only if s1 and s2 contain the same
//       keys in the same order
bool operator== (const stack& s1, const stack& s2);
```

A main-function to test the new stack operations is available in extended_stack_template.cpp.

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

```
Input numbers: -1 2 -3 4
stack == copy ? 1
stack == modified ? 0
size ? 4
size modified ? 3
```

```
Input numbers: 1
stack == copy ? 1
stack == modified ? 0
size ? 1
size modified ? 0
```

Submission: https://challenge.inf.ethz.ch/team/websubmit.php?cid=5&problem=MP15131

Assignment 2 – Skript-Aufgabe 160 (4 points)

A *queue* is a data structure whose core functionality consists of the following two operations:

1. Insertion of a new element at the end ("element queues up");
2. Removal of the first element ("element gets served").

Implement a type `ifmp::queue` that supports these operations, with the following public member functions (and keys of type `int`).

```
// POST: key is added as last element
void push_back (int key);

// POST: returns whether *this is empty
bool empty () const;

// PRE: !empty()
// POST: first element of *this is returned
int front () const;

// PRE: !empty()
// POST: last element of *this is returned
int back () const;

// PRE: !empty()
// POST: first element of *this is removed
void pop_front ();
```

In addition, as a queue is a dynamic type, you must also provide copy constructor, assignment operator, and destructor. Furthermore, `queue_template.cpp` provides a `main`-function to test your implementation. It pushes an integer n to a queue when `a` n is entered and returns and pops the first element when `b` is entered. There are examples in the judge box below.

Hint: Queues work similarly to stacks, thus you can take the codes for `ifmp::stack` as a reference. But notice that in order to submit to the judge you must correctly pack everything into one file.

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

```
a -1 a 2 a -3 a 4 b b
-1 2
Last element: 4
```

```
a 1 b
1
Last element: none (queue empty)
```

```

Last element: none (queue empty)
```

Submission: https://challenge.inf.ethz.ch/team/websubmit.php?cid=5&problem=MP15132

Assignment 3 – (4 points)

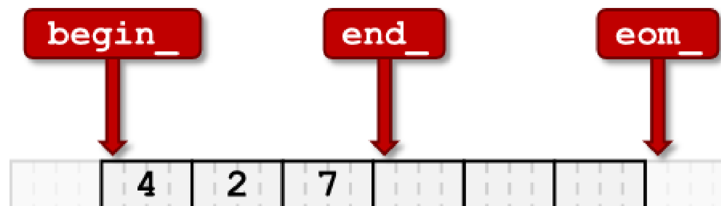
Note: It's the same exercise as on sheet 12. The submission deadline was extended by one week.

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