ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecturer: Prof. B. Gärtner

Informatik für Mathematiker und Physiker HS13

Exercise Sheet 13

Submission deadline: 15:15 - Tuesday 17th December, 2013
Course URL: http://www.ti.inf.ethz.ch/ew/courses/Info1_13/

Assignment 1 (4 points)

The fact that an array has fixed length is often inconvenient. For example, in Exercise 2 of Homework 6, the number of elements to be read into the vector had to be provided as the first input in order for the program to be able to dynamically allocate a vector of the appropriate length (see `read_array.cpp`). Now we would like to write a program that reads a sequence of integers from standard input into an array, where the length of the sequence is not known beforehand (and not part of the input)—the program should simply read one number after another until the stream becomes empty.

One possible strategy is to dynamically allocate an array of large length, big enough to store any possible input sequence. But if the sequence is short, this is a huge waste of memory, and if the sequence is very long, the array might still not be large enough.

a) Write a program `read_array2.cpp` that reads a sequence of integers of unknown length into an array, and then outputs the sequence. The program should satisfy the following two properties.

   (i) The amount of dynamically allocated memory in use by the program should at any time be *proportional to the number of sequence elements that have been read so far*. To be concrete: there must be a positive constant a such that no more than ak cells of dynamically allocated memory are in use when k elements have been read, $k \geq 1$. We refer to this property as *space efficiency*. It ensures that even very long sequences can be read (up to the applicable memory limits), but that short sequences consume only little memory.

   (ii) The number of assignments (of values to array elements) performed so far should at any time be proportional to the number of sequence elements that have been read so far, with the same meaning of proportionality as above. We refer to this property as *time efficiency*. It ensures that the program is only by a constant factor slower than the program `read_array.cpp` that knows the sequence length in advance.

b) Determine the constants of proportionality a for properties (i) and (ii) of your program.

Note: You must not use any data structure from the STL library (like for instance `std::vector`) to accomplish task a), you should rather use `new` and `delete` to dynamically allocate arrays.

Assignment 2 (6 points)

A `std::vector` is a container that represents arrays that can change in size. 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.

In this exercise we ask you to implement a class `ifmp::vector` that provides *similar* functionality as the class `std::vector`, at least as far as dynamic memory management is concerned.

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. Therefore you should use here the same technique that you developed in the previous exercise for the program `read_array2.cpp`.

Download the files `vector.h`, `vector.cpp` from the webpage and implement the member functions of `ifmp::vector`. Of course, as in the previous exercise, you are not allowed to use any data structure from the STL 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.