An Approach for Formalizing the Memory Consumption of C++ Standard Template Library Containers

Attila Gyen, Norbert Pataki

Department of Programming Languages and Compilers, Eötvös Loránd University

gyenattila@gmail.com, patakino@elte.hu

The Standard Template Library [1] (STL) which initially developed by Hewlett-Packard Company in 1994 became the library for the standard C++ language. In the following years, this was significantly expanded and enriched with many new elements in the C++11 / C++14 standards. The template library supports generic programming with containers, iterators (generalized pointers), and algorithms [2]. The template solution allows us to use the classes and functions with a given name for (almost) any type, according to the needs of the program. Containers have different characteristics in many ways:

- time required to insert or delete a new item
- access time to stored items
- regarding the memory usage, containers can be divided into two main groups: memory-contiguous and node-based

In STL, there is an asymptotic run-time guarantee for most library algorithm operations that are performed on containers, but there is less conversation about how container's memory is allocated or re-allocated on the heap. After all, a vector and a linked list have a different memory representation, such as a double-ended queue or a map. With n number of elements, we can be sure that these data structures will need different amounts of memory allocation even if they are of the same type of elements.

It is almost impossible to make an exact prediction because the size of each type is not known in advance, especially the user-defined types. However, it is possible to determine memory requirements in advance, such as that the size of an x container containing integers of n elements is sure to always be less than or equal to that of a y container, where x and y are different types of containers. In order to provide an even more generic solution, we disregard the platform-dependent memory allocation of C++ primitive types.

The purpose of this paper is to provide a detailed explanation of these guarantees. We present an approach how to formalize the heap memory requirements for certain containers.

References

[1] Norbert Pataki, C++ Standard Template Library by template specialized containers Acta Univ. Sapientiae, Inform. 3,2 (2011) 141–157

[2] Matthew H. Austern, *Generic programming and the STL: using and extending the C++ Standard Template Library*, 1998