Description of C++ examples in cppexamples.zip

By Agner Fog, © 2008-2022.
Available at www.agner.org/optimize/cppexamples.zip.
Discussed in www.agner.org/optimize/optimizing_cpp.pdf.

Overview of container classes and templates

Container	Maximum size set at runtime	Can grow	Can remove objects	Can insert/delete at random positions	Elements can have constructors/destructors	Can sort elements	Can search for elements	Random access by index	Allows First-In-First-Out access	Allows First-In-Last-Out access	Can contain objects of mixed type	Can contain strings of different lengths	Memory is contiguous	Comments
In cppexamples.zip:														
SafeArray	-	-	-	-	-	-	-	X	-	-	-	-	X	Linear array
FIFOlist	-	X	X	-	-	-	-	-	X	-	-	-	X	Circular array
FILOlist	-	X	X	-	-	-	-	-	-	X	-	-	X	Stack
SortedList	-	X	X	X	-	X	X	X	-	-	-	-	X	Sorted list
DynamicArray	X	X	X	-	-	-	-	X	-	X	-	-	X	Linear array
DynamicQueue	X	X	X	-	-	-	-	X	X	-	-	-	X	Circular array
MixedPool	X	X	X	-	-	-	-	X	-	-	X	-	X	Memory pool
StringPoolS	X	X	X	-	-	-	-	X	-	-	-	X	X	String memory pool
StringPoolL	X	X	X	-	-	-	-	X	-	-	-	X	X	String memory pool
StringPoolW	X	X	X	-	-	-	-	X	-	-	-	X	X	String memory pool
Standard templates:														
STL vector	X	X	-	X	X	-	-	X	-	X	-	-	X	Linear array
STL list	X	X	X	X	X	X	-	-	X	X	-	-	-	Doubly linked list
STL deque	X	X	X	X	X	-	-	-	X	X	-	-	-	Double ended queue
STL set, multiset	X	X	X	X	X	X	X	X	-	-	-	-	-	Binary tree
STL map, multimap	X	X	X	X	X	X	X	X	-	-	-	-	-	Binary tree
STL hash_map, unordered_map	X	X	X	X	X	-	X	X	-	-	-	-	-	Hash map

The above table shows an overview of container classes provided in cppexamples.zip as well as standard C++ containers (previously known as the Standard Template Library, STL) for comparison.

Explanation of the table headings:

Maximum size set at runtime:
The maximum size of the container, i.e. the number of objects or elements it can contain, does not have to be known at compile time but can be set at runtime depending on the data.

Can grow:
The size of the container can be increased at any time. The final size does not have to be known before the first element is added.

Can remove objects:
Objects can be removed from the container.

Can insert/delete at random positions:
Objects can be added or removed not only at the end of a list, but at any position in the sequence.

Elements can have constructors/destructors:
Can contain objects of a class that has non-default constructors, copy constructors, move constructors, or destructors. A '-' in this column means that any constructors/destructors of the objects stored are not called correctly when individual objects are moved, copied or destroyed. The container itself does have constructor and destructor.

Can sort elements:
Elements added in random order can be sorted.

Can search for elements:
A search feature makes it possible to find an object with a particular value or key without looking through the entire list.

Random access by index:
Each element can be accessed by a unique index or key, typically of the form `list[i]`.

Allows First-In-First-Out access:
Can be used as a queue with First-In-First-Out access.

Allows First-In-Last-Out access:
Can be used as a stack with First-In-Last-Out access.

Can contain objects of mixed type:
Can be used as a memory pool storing objects of different types in the same memory block.

Can contain strings of different lengths:
Useful for storing text strings.

Memory is contiguous:
Objects are stored contiguously in one big memory block rather than being spread among multiple memory blocks. This can reduce memory fragmentation and heap overhead, and improve cache efficiency.

Explanation of each container class or template:

SafeArray:
A simple array with bounds checking. Size defined at compile time.

FIFOlist:
A First-In-First-Out queue. Size defined at compile time.

FILOlist:
A First-In-Last-Out stack. Size defined at compile time.

SortedList:
A list that is kept sorted at all times. Objects can be added or removed in random order. Objects can be found by binary search. Size defined at compile time. This is efficient for small lists but inefficient for large lists. Use a binary tree or hash map instead if the list is very large.

DynamicArray:
A linear array that can be resized at any time. Can also be used as First-In-Last-Out stack.

DynamicQueue:
A First-In-First-Out queue. Size defined at runtime.

MixedPool:
A memory pool for storing objects of different types in the same contiguous memory block. This can reduce memory fragmentation and heap overhead and improve cache efficiency. Can also be used for reading and writing binary files containing mixed data structures.

StringPoolS:
A memory pool for storing zero-terminated text strings (ASCII or UTF-8) together in the same contiguous memory block. Includes various string manipulation functions.

StringPoolL:
Same as StringPoolS. The length of each string is stored in order to make string manipulation faster.

StringPoolW:
Same as StringPoolS, but for 16-bit characters (`wchar_t`).

Further container class templates suitable for handling vectors and matrixes are provided at https://github.com/vectorclass/add-on/tree/master/containers

Why not use standard templates?

The standard C++ container class templates (previously known as the standard template library, STL) are certainly useful for many programmers. The advantages are that they are flexible, standardized, and well tested. If a container fulfills your needs then use it!

However, the standard containers are designed for generality and flexibility, while execution speed, heap efficiency, cache efficiency and code size have got low priority. The container classes and templates in cppexamples.zip, on the contrary, are designed with more focus on code efficiency. In particular, the number of memory allocations and re-allocations is kept at a minimum by keeping multiple objects together in the same memory pool and by growing the memory blocks by large amounts when they are exhausted, where the standard containers typically grow memory blocks in smaller steps.

The code size of each example is kept so small that a proficient C++ programmer will be able to understand it and to modify it if desired.

The container classes and templates provided here in cppexamples.zip can be used instead of standard containers when fast execution, small code, and efficient caching are of prime importance.

These containers do not use structured exception handling because general performance can be improved by compiling without support for exception handling. Instead, they write error messages to the standard error output. You may modify this behavior to provide a pop-up box that fits a specific graphical user interface or a general error handling strategy or your application.