C++ Tidbits

Gokul Nadathur

gokul.nadathur@gmail.com

Virtual

- Virtual destructors of base classes ensure that the destructor of the inherited is called when appropriate.
- Virtual functions are implemented using an array of function pointers per class. This is called the `vtable`. A member variable `vptr` is added to the class by the compiler. `vptr` points to `vtable`. Code is generated in every constructor to initialize `vptr` when an object is created. `dynamic_cast` uses this table to figure out type of the object in runtime.
- Keyword **Override**: Added to declaration of the inherited function. Generates a build error if the signature of the virtual base function diverges from the signature of the inherited.

Exceptions

The compiler maintains exception handling in side tables which are not consulted unless an exception needs to be handled. The 2 main data structures used in handling exceptions are:

- `.eh_frame`: It contains the information (DWARF) that is required to pop back to the state of the machine registers and the stack at any point higher up the call stack.
- `.gcc_except_table`: contains information that is necessary in order to know when to stop unwinding the stack.

Performance Aspects: Exceptions are slow because:

1. when an exception occurs, the function call stack is linearly searched for the exception handler, and all the entries before the function with exception handler are removed from the function call stack.
2. Temporary exception objects get created and destroyed as the stack is unwound. To avoid latter overhead, exception objects are caught as constant reference.

STL Container Choice Schema

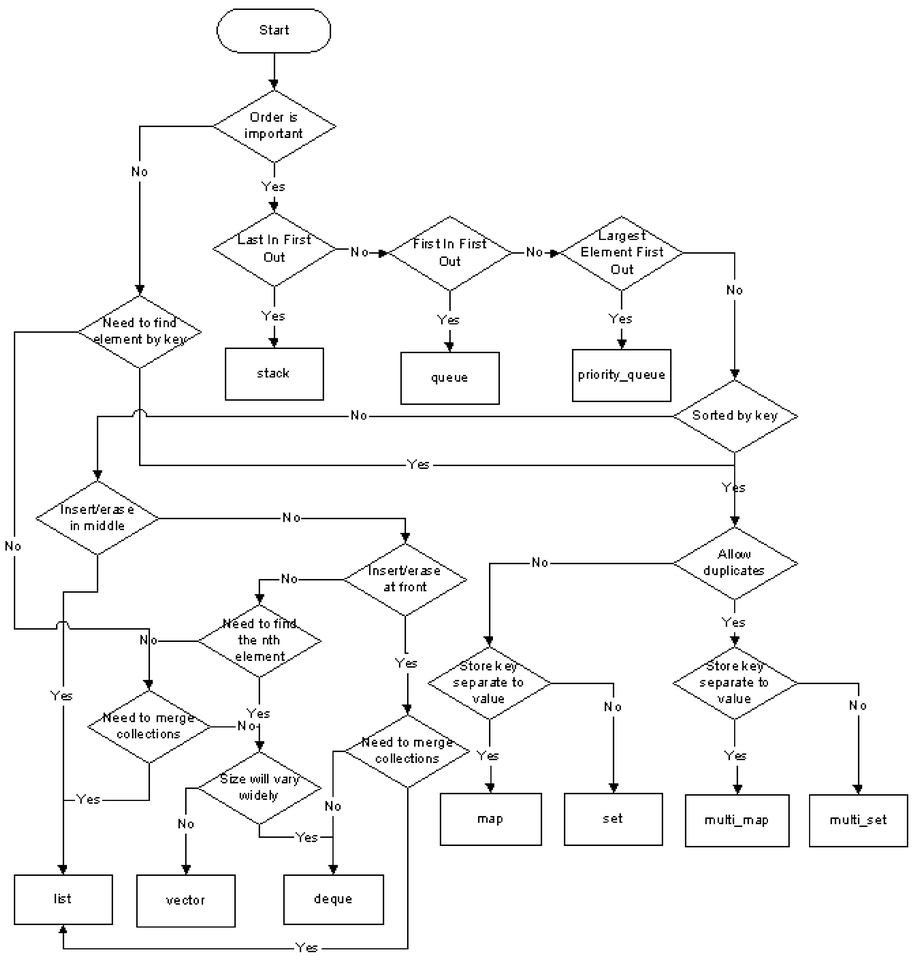

Figure 1: Container Choice Flowchart

STL tidbits

`list` iterator to an element in the list is safe from mutation of the list itself unless the element pointed to by the iterator is erased. `emplace(_*)` APIs are faster than `push(_*)` APIs when constructing objects as a part of the call. `emplace(_*)` constructs the object in-place rather than creating a temporary copy of the object on the stack.

Smart Pointers

- `unique_ptr`: Destroys the object when the pointer goes out of scope. Ownership can be transferred but only 1 pointer is valid.
- `shared_ptr`: Provide a reference counted access to the allocated object. Object is destroyed only when all references are invalid.
- `weak_ptr`: To avoid reference cycles with `shared_ptr`, `weak_ptr` contains the pointer to the object corresponding to the `shared_ptr` but does not participate in the reference counting.

Miscellaneous

Friendship

- `friend` class and function can access private members of class.
- `friend` function can be member function of another class or global function.
- Friendship is not inherited.