STL - Grunnleggende

Standard Library i C++ består av fire hoveddeler:

1. C standardbiblioteket
   - cstdio, cstring, cstdbool, cstdlib, cctype, ctime, cmath, …..

2. I/O stream bibliotek
   - iostream, fstream, iomanip, …..

3. Bibliotek med diverse
   - string, memoryhåndtering, exceptions, …..

4. **Standard Template Library (STL)**
   - Containers (vector, list, map, stack, queue, array, set, …..)
   - Iterators
   - Algorithms (sort, find, copy, merge, move, swap, replace, …..)

Container:

- **er *et objekt* som i hukommelsen lagrer andre objekter/elementer!**
  Altså lagre unna datastrukturer i container(e)!

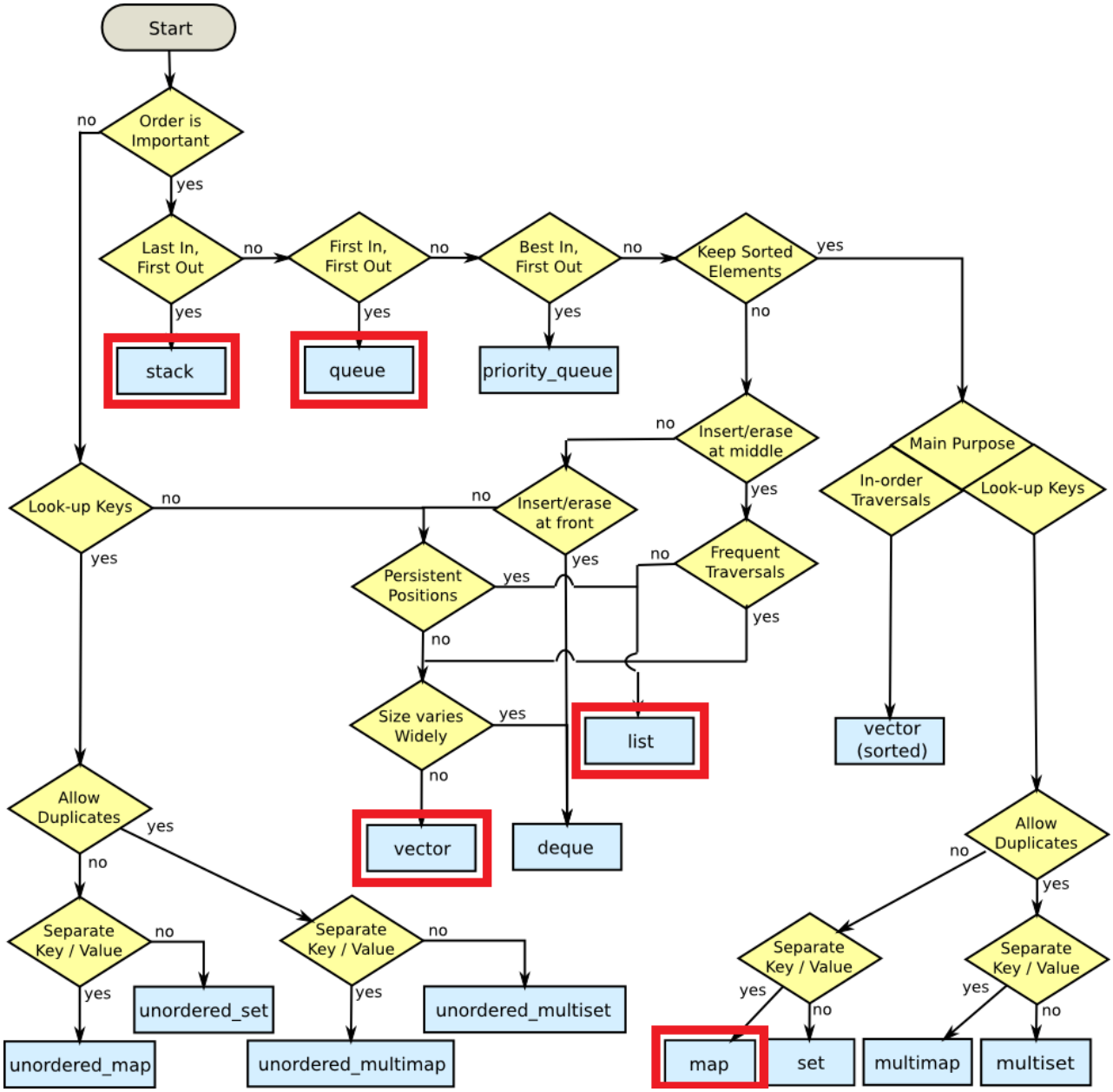
- **lagrer (kopi) av verdier/variable – *ikke* referanser**
  NB: Om da det ikke er *pekere som lagres* – jfr. mange av eksemplene med `vector` hittil. Vi skal i dette emnet derfor *ikke lagre objekter* i containerne, men *pekere* til slike. Å sortere slike tilpekte objekter er da litt verre. Men, langt fra umulig. Vi skal bl.a. bruke noe som heter lambda-funksjoner til dette (jfr.kap.11.8 og EKS_25.CPP). Til gjengjeld slipper vi bl.a. å måtte lage en egen copy-constructor eller overloading av operatoren '<' (heller ikke lært noe om, jfr. 11.6) inni mange/alle av klassene.

- inndeles i tre typer:
  - **sekvens**: ***vector, list***, array, deque, forward_list
    Bruker ofte *indeks* (ved vector) for å aksessere elementene.
  - **assosiative**: ***map***, set, multimap, multiset, bitset, unordered_…….
    Bruker ofte *nøkkel/verdi* for å aksessere elementene.
  - **adapters**: ***stack, queue***, priority_queue
    Bruker ofte *posisjonen (foran/bak)* for å aksessere elementene.

- ***må* ha: #include <……>** (string, vector, list, stack, queue, map)

***Mange* ulike containere i STL** (*ikke fullt alle* er med på figuren)**:**

(**men:** vi skal i vårt emne *kun* se på/bruke de fem med røde rammer rundt)



Kilder: https://ngoduyhoa.blogspot.com/2015/06/summary-of-different-containers.html
https://stackoverflow.com/questions/471432/in-which-scenario-do-i-use-a-particular-stl-container
https://i.stack.imgur.com/G70oT.png

Våre 4x containere (i tillegg til `vector`):

- **List: mye likt med `vector`, men:**
  - Kan *ikke* aksessere element vha. indeks
  - Enkelt å holde elementene sortert etter en nøkkel/verdi
  - Enkelt/raskt å fjerne/sette inn element hvor som helst
  - Raskt og enkelt flette sammen to *sorterte* lister
  - Egne funksjoner for å sette inn/ta ut noe også aller forrest
  - Enkelt fjerne alle duplikater
  - Toveis liste (kan gå både forover og bakover i lista)

- **Stack:**
  - Brukes når man *kun* skal sette inn og ta ut elementer aller ***forrest***
  - Kan implementeres også som en `vector`, `list` (eller `deque`)

- **Queue:**
  - Brukes når man hele tiden *kun* skal sette inn elementer aller ***bakerst*** og bestandig ta ut elementer aller ***forrest***
  - Kan implementeres også som en `vector`, `list` (eller `deque`)

- **Map:**
  - *Unike nøkler/verdier med tilhørende data* legges *parvis* inn
  - Lagres stigende sortert
  - Skal *vi* bruke mest ifm. *sorterte unike objekter det er brukt pekere til*

En del sentrale funksjoner:

Funksjoner/metoder felles for disse fire:

size, begin, end, rbegin, rend, empty, clear, erase, insert, swap, emplace

- **vector**: [ ], push_back, pop_back, at, front, back, capacity, ……
- **list**: push_back, push_front, pop_back, pop_front, front, back, remove, sort, reverse, merge, unique….
- **map**: [ ], find, count, lower_bound, upper_bound, ……
- **string:** [ ], length, push_back, at, append, find…, replace, substr, ……

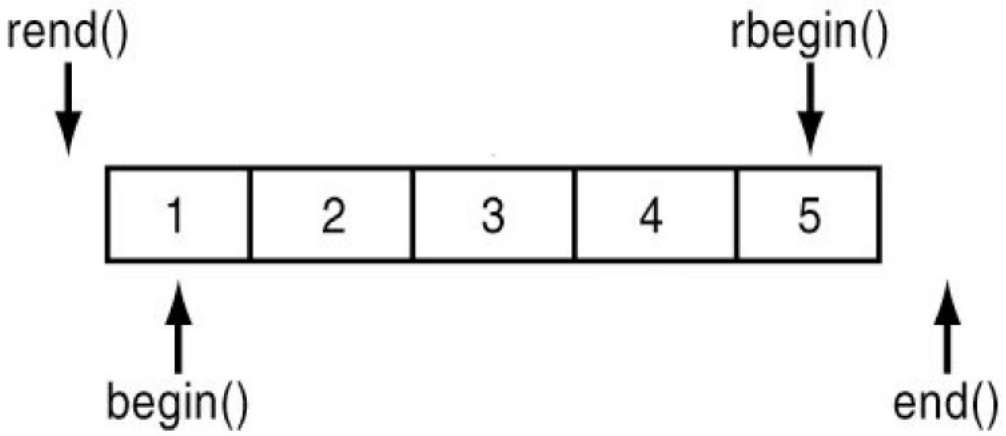
Funksjoner/metoder felles for begge: empty, size, push, pop, swap, emplace

- **stack**: top
- **queue**: front, back

Se EKS_21.CPP og EKS_22.CPP

Iterator:

- Mange containere (bl.a. *ikke* `stack` og `queue`) har en tilhørende *iterator*-klasse som kan brukes til å gå gjennom (alle) elementene i containeren
- Iteratoren er *en peker*
- Denne går *f.o.m* `begin` og *opp til* `end` (men *ikke* t.o.m. `end`)



- kan gjøre null, en eller flere av:
  - ++, --
  - indekseres: [ ]
  - dereferenseres på høyre/venstre side (…. = *it *it = …..)
  - brukes sammen med operatorer: == != + - += -= < <= > >=
  - kopieres
- Er *ikke sjeldent* udefinert («søppel») etter at brukt i en operasjon/funksjon!

Se EKS_23.CPP og EKS_24.CPP

Algorithm:

- Filen <algorithm> inneholder ca.100 funksjoner for å håndtere innholdet i containere. *Dette er generelle funksjoner som **ikke** ligger inni/tilhører en spesiell container*. De aller fleste av dem tar i stedet bl.a. inn som parametre en start-iterator og en slutt-iterator. Funksjonen operere så på alle elementene *f.o.m*. den første iteratoren og *opp til* den andre. Mange av funksjonene tar også en annen funksjon som parameter! Dette *må* gjøres for å kunne utføre de rette operasjoner på den aktuelle og konkrete datatypen.

- Funksjonene *kan* f.eks. deles inn i disse fem kategoriene:
  - **Nonmutating algorithms**: Nonmutating algorithms are the algorithms that do not alter any value of a container object nor do they change the order of the elements in which they appear. These algorithms can be used for all the container objects, and they make use of the forward iterators.

- **Mutating algorithms**: Mutating algorithms are the algorithms that can be used to alter the value of a container. They can also be used to change the order of the elements in which they appear.
- **Sorting algorithms**: Sorting algorithms are the modifying algorithms used to sort the elements in a container.
- **Set algorithms**: Set algorithms are also known as sorted range algorithm. This algorithm is used to perform some function on a container that greatly improves the efficiency of a program.
- **Relational algorithms**: Relational algorithms are the algorithms used to work on the numerical data. They are mainly designed to perform the mathematical operations to all the elements in a container.

(Hentet fra: https://www.javatpoint.com/cpp-stl-components)

- *Noen få* eksempler på slike algoritmer:
  - `n = count(iter1, iter2, verdi)` – returnerer antall ganger `verdi` forekommer mellom iteratorene
  - `for_each(iter1, iter2, funk)` – kjører/tilkaller funksjonen `funk` for/inni hver av elementene
  - `iter3 = max_element(iter1, iter2)` – returnerer peker til den *største*
  - `copy(iter1, iter2, iter3)` – kopierer elementene til der `iter3` peker (som kan hende er i en annen container!) *Overskriver* elementer – *inserter* ikke!!!
  - `replace(iter1, iter2, gmlVerdi, nyVerdi)` – *alle* verdier lik `gmlVerdi` erstattes med `nyVerdi`
  - `iter5 = search(iter1, iter2, iter3, iter4)` – returnerer (om mulig) peker til der elementene mellom `iter1` og `iter2` forekommer i intervallet `iter3` til `iter4` (der intervallene gjerne er i ulike containere)

- Ellers er flere funksjoner med samme navn og virkemåte som inni containerne også tilgjengelig (uten å måtte angi containerens navn, men bare iteratorene), f.eks: `find`, `merge`, `remove`, `reverse`, `sort`, `swap`

Se EKS_25.CPP

Se mange ulike linker til *alt* dette under:
«String-klassen, Containers og Algorithms» og
«Quick Reference Cards/Cheat Sheets»
på «Ressurser/mer stoff»-siden på emnets hjemmeside.