Контейнеры STL

Последовательные:
```
vector<T>
list<T>
forward_list<T>
deque<T>
```

Ассоциативные:
```
map<K,V>
multimap<K,V>
set<K>
multiset<K>
unordered_map<K,V>
unordered_multimap<K,V>
unordered_set<K>
unordered_multiset<K>
```

Адаптеры контейнеров:
```
stack<T>
queue<T>
```

Общие контейнерные члены

```
container<T>::iterator
container<T>::const_iterator
container<T>::size_type
container<T>::value_type - синоним для T
container<T>::difference_type – тип разности между итераторами
```

Общие контейнерные операции

```
container<T> c {};
container<T> c {elems};
container<T> c {c1};
container<T> c {move(c1)};
container<T> c {b,e} – инициализация значениями между парой итераторов
c = c1                - присваивание (для элементов вызывается operator=)
c = {elems}
c = move(c1)
c.assign(b,e)         - замена содержимого контейнера элементами диапазона [b,e)

c.begin()           - итератор начала
c.end()             - итератор конца
c.cbegin()          - константный итератор начала
c.cend()            - константный итератор конца
c.rbegin()          - обратный итератор начала
c.rend()            - обратный итератор начала
c.size()            - размер
c.empty()           - проверка на пустоту
c.clear()           - очистка
c.swap(c1)          - обмен местами элементов двух однотипных контейнеров
c==c1               - проверка на равенство
c!=c1               - проверка на неравенство
c<c1  >  <=  >=     - лексикографическое сравнение
```

Операции, характерные для последовательных контейнеров

```
container<T> c(n)   – контейнер из n элементов со значением по умолчанию
container<T> c(n,t) – контейнер из n элементов, заполненных значением t

c.assign(n,t)       - присваивание n копий t

c.push_back(t)      - добавление в конец
c.emplace_back(args)- построение объекта типа T из аргументов и добавление его в конец
                      c.push_back(Person("Иванов",20)) ~ c.emplace_back("Иванов",20)
c.pop_back()        - удаление последнего элемента (возвращает void)

c.insert(it,t)      - вставка элементов в контейнер перед итератором it
c.emplace(it,args)  - построение объекта типа T из аргументов и вставка перед итератором it
c.insert(it,n,t)      Для vector все итераторы этого контейнера
c.insert(it,{elems})  становятся недействительными
c.insert(it,b,e)      Возвращают итератор вставленного элемента


c.erase(it)         - удаление элемента, на который ссылается it
c.erase(b,e)        - удаление элементов в диапазоне [b,e)
                      Возвращают итератор элемента, следующего за удаленным

c.front()           - ссылка на первый элемент
c.back()            - ссылка на последний элемент
```

Операции, характерные только для vector и deque

```
c[i]                - произвольный доступ без проверки
c.at(i)             - произвольный доступ с проверкой (исключение out_of_range)
```

Операции, характерные только для vector

```
c.resize(n)         - изменение размера вектора
c.capacity()        - емкость вектора
c.reserve(n)        - резервирование емкости
c.shrink_to_fit()   - уменьшение емкости до размера
```

Операции, характерные для списков

```
l.push_front(t);       - добавление элемента в начало
l.emplace_front(args); - построение объекта типа T из аргументов и добавление его в
                         начало
l.pop_front();         - удаление первого элемента

l.sort()               - специализированная сортировка списков
l.sort(Cmp)

l.merge(l1)            - объединение двух сортированных списков с удалением
                       - элементов из l1 и вставкой в l с сохранением порядка

l.remove(t)            - удаление всех элементов t
l.remove_if(Pred)      - удаление всех элементов, удовлетворяющих предикату Pred
l.reverse()

l.unique()             - удаление рядом стоящих дубликатов, используя ==
l.unique(BinPred)      - удаление рядом стоящих дубликатов, используя BinPred

l.splice(it,l1)        - перемещение элементов из списка l1 в список l перед it
l.splice(it,l1,it1)    - перемещение элемента *it1 из списка l1 в список l перед it
l.splice(it,l1,b,e)    - перемещение элементов [b,e) из списка l1 в список l перед it
```

Стек, очередь

```
stack<T> s;
stack<T,vector<T>> s1;
s.push(x)
void s.pop()
s.top()
s.size()
s.empty()
```

```
queue<T> q;
queue<T,list<T>> s1;
q.push(x)
void q.pop()
q.top()
q.size()
q.empty()
```

Лямбда-выражения

```
auto f = [](int x){ return x*x; };
auto f = [](auto x){ return x*x; };
cout << f(2);
[](int x) -> double { return x*x; }
[a](int x) { return x<a; }
[this](int x) { return x<field; } // мы – в методе класса с полем field
[this]() { this->MemberFunction(); }
[&s](int x) { s += x; }
```

Тип лямбда-выражения

```
#include <functional>
std::function<int(int)> f = [](int x) { return x*x; };
```

Виды списков захвата

```
[a,&b] [&] [=] [=, &y]
```