Vector

1. Какие виды векторов позволяют создать конструкторы?
   - Пустой
   - С n элементами, где каждый элемент - это копия заданного значения
   - Вектор, заполненный значениями из диапазона [first, last)
   - Копию вектора

```
std::vector<int> first;                                // empty vector of ints
std::vector<int> second (4,100);                       // four ints with value 100
std::vector<int> third (second.begin(),second.end());  // iterating through second
std::vector<int> fourth (third);                       // a copy of third
```

2. Как задать размер контейнера?

```
std::vector<int> vector;
vector.resize(5);
```

3. Как определить потенциальный размер контейнера?

```
std::vector<int> vector;
vector.capacity();
```

4. Как получить доступ к элементу контейнера?

```
std::vector<int> vector;
vector[5];
vector.at(5);
```

5. Как присвоить значение элементу контейнера?

```
std::vector<int> vector;
vector[5] = 3;
vector.at(5) = 3;
```

6. Опишите функциональность методов begin и end
   - begin - возвращает итератор, указывающий на первый элемент вектора
   - end - возвращает итератор, относящийся к “после последнему” теоретическому элементу вектора

```
std::vector<int> vector;
for (std::vector<int>::iterator it = vector.begin() ; it != vector.end(); ++it)
    std::cout << ' ' << *it;
std::cout << '\n';
```

7. Опишите использование метода back и front
   - front- возвразает ссылку на первый элемент
   - back - возвразает ссылку на последний элемент

```
std::vector<int> vector;
vector.front();
vector.back();
```

8. Опишите использование метода insert
   - используется для вставки новых элементов (нового элемента) перед элементом в указанной позиции

```
std::vector<int> vector;
std::vector<int> anothervector;
vector.insert (vector.begin(), 3);
vector.insert (vector.begin()+1, 2, 5);
vector.insert (vector.begin()+2, anothervector.begin(),anothervector.end());
```

9. Как и куда можно вставить новые элементы в контейнер?
   - в любое место контейнера с помощью push_back, insert, emplace

```
std::vector<int> vector;
vector.push_back(5);
vector.insert (vector.begin(), 3);
vector.emplace (vector.begin()+1, 4);
```

10. Опишите функциональность методов capacity и max_size

- capacity - возвращает потенциальный размер вектора
- size - возвращает количество элементов в векторе

```
std::vector<int> vector;
vector.capacity();
vector.size();
```

11. Зачем нужны методы reserve и resize?
    - reserve - измеяет capacity
    - resize - изменяте размер вектора

```
std::vector<int> vector;
vector.resize(5);
vector.reserve(100);
```

12. Как вставит/удалить элемент в начало контейнера?

```
std::vector<int> vector;
vector.insert (vector.begin(), 4);
vector.emplace (vector.begin(), 5);
```

13. Как вставит/удалить элемент в конец контейнера

```
std::vector<int> vector;
vector.push_back(5);
```

Deque

14. Какие виды двусторонних очередей позволяют создать конструкторы?
    - Пустая
    - С n элементами, где каждый элемент - это копия заданного значения
    - Очередь, заполненная значениями из диапазона [first, last)
    - Копию очереди

```
std::deque<int> first;
std::deque<int> second (4,100);
std::deque<int> third (second.begin(),second.end());
std::deque<int> fourth (third);
```

15. Как получить доступ к элементу контейнера?

```
std::deque<int> deque;
deque.at(5);
deque[5];
```

16. Как присвоить значение элементу контейнера?

```
std::deque<int> deque;
deque.at(5) = 3;
deque[5] = 3;
```

17. Опишите функциональность методов front и back
    - front- возвразает ссылку на первый элемент
    - back - возвразает ссылку на последний элемент

```
std::deque<int> deque;
deque.front();
deque.back();
```

18. Как вставит/удалить элемент в начало контейнера?

```
std::deque<int> deque;
deque.emplace_front(1);
```

19. Как вставит/удалить элемент в конец контейнера

```
std::deque<int> deque;
deque.emplace_back(1);
```

List

20. Опишите использование метода merge

- A.merge(B) - перемещает все элементы из B в A на соответствующие упорядоченные позиции

```
int ints1[]= {1, 3, 5};
int ints2[]= {2, 4, 6};
std::list<int> list1(ints1,ints1+3);
std::list<int> list2(ints2, ints2+3);
list1.merge(list2);
```

21. Опишите использование метода remove_if

- удаляет из листа элементы, для которых предикат возвращает значение истинна

```
bool less_than_ten (const int& value) { return (value<10); }
int main()
{
    int myints[]= {15,36,7,17,20,39,4,1};
    std::list<int> list(myints,myints+8);
    list.remove_if (less_than_ten);
```

22. Опишите использование метода splice

- A.splice(iterator pos, list& B) - перемещает элементы из B в A в указанную позицию

```
int ints1[]= {1, 2, 6};
int ints2[]= {3, 4, 5};
std::list<int> list1(ints1,ints1+3);
std::list<int> list2(ints2, ints2+3);
list1.splice(++(++list1.begin()), list2);
```

23. Опишите использование метода unique

- в каждой последовательной группе равных элементов, удаляет все элементы, кроме первого.

```
int ints1[]= {1,2,2,6,6,6,1,2,4,4};
std::list<int> list(ints1,ints1+10);
list.unique();
// list: 1 2 6 1 2 4
```

Stack

24. Опишите функциональность методов push и top и как она реализуется в зависимости от выбора базового контейнера?

- push - вставляет новый элемент в вершину стека
- top - возвращает ссылку на вершину стека
- push и top вызывают методы push_back и back базового контейнера

```
std::stack<int> stack;
stack.push(10);
stack.top();
```

25. Почему стек реализован как адаптер контейнера? Каким образом?

- потому что проще было написать через готовый контейнер
- стек использует методы базового класса и организован по принципу “последним зашел - первым вышел”

26. Каким образом изменить умолчание в выборе базового контейнера?

```
std::stack<int, std::vector<int> > stackBasedOnVector;
```

Queue

27. Опишите функциональность методов push и front и как она реализуется в зависимости от выбора базового контейнера?

- push - вставляет новый элемент в конец очереди
- front- возвращает ссылку на первый элемент очереди
- push и front вызывают методы push_back и back базового контейнера

```
std::queue<int> queue;
queue.push(77);
queue.front();
```

28. Почему очередь реализован как адаптер контейнера? Каким образом?

- потому что проще было написать через готовый контейнер
- очередь использует методы базового класса и организована по принципу “первым зашел - первым вышел”

29. Каким образом изменить умолчание в выборе базового контейнера?

```
std::queue<int, std::list<int> > queue;
```

Общие вопросы

30. У каких контейнеров допустим произвольный доступ к элементам?

- vector
- deque
- array

```
vector.at(2);
vector[2];
deque.at(2);
deque[2];
array.at(2);
array[2];
```

31. Для каких контейнеров сохраняются значения указателей, итераторов после вставки/удаления?

- vector
- deque
- list
- forward_list

```
std::vector<int>::iterator itVector = vector.insert(vector.begin(), 200);
std::deque<int>::iterator itDeque = deque.insert(deque.begin(), 200);
std::list<int>::iterator itList = list.insert(list.begin(), 200);
std::forward_list<int>::iterator itForwardList = forward_list.insert_after(forward_list.begin(), 200);
```

32. Какие последовательные контейнеры поддерживают упорядоченность элементов автоматически?

- map
- set

```
map<char,int> map;
set<int> set;

set.insert(10);
set.insert(20);
set.insert(15);

map.insert(std::pair<char,int>('a',100));
map.insert(std::pair<char,int>('z',200));
map.insert(std::pair<char,int>('m',150));
```

33. Какие средства можно использовать для сортировки элементов контейнера?

- можно использовать sort из библиотеки algorithm

```
sort(container.begin(), container.end());
```

34. Когда, в каких случаях нужно отдать предпочтение выбору одного из контейнеров?

- выбор контейнера зависит от конкретной задачи. Так, например, если в задаче требуется только вставка элементов, целосообразно использовать list, тк вставка работает за O(1)

35. Какие средства можно использовать и что необходимо реализовать для сравнения элементов контейнеров, если они содержат объекты пользовательских классов?

- можно использовать операторы >, <, ==, !=, <=, >=. Для этого надо перегрузить их для пользовательского класса

```
vector<A> vectorA;
A a1(3);
A a2(3);
vectorA.push_back(a1);
vectorA.push_back(a2);
cout << (vectorA[0] == vectorA[1]); // true
```