1. **Какой оператор следует использовать для определения типа объекта во время работы программы?**

====

#typeid

++++

2. **Какой заголовочный файл нужно подключить для определения типа объекта во время работы программы?**

====

#typeinfo

====

++++

3. **Укажите строку, в которой записан член класса, определенный в классе type_info.**

====

#bool operator!=(const type_info &ob);

====

4. **Укажите строку, в которой записан член класса, определенный в классе type_info.**

====

#const char *name();

++++

5. **Какие операции сравнения перезагружены в классе type_info?**

====

== и != .

====

6. **операторы dynamic_cast ...**

====

#исползуются для приведения полиморфных видов во время работы программы

====

++++

7. **операторы const_cast ...**

====

#исползуются для явного переопределения модификаторов const и / или volatile

====

++++

8. **операторы static_cast ...**

====

#исползуются для конвертации одного фундаментального типа данных в другой

====

9. **операторы reinterpret_cast** ...

====

#позволяет преобразовывать любой целочисленный тип в любой тип указателя и наоборот

++++

namespace - это ...

====

#это пространство имен, которое решает проблему использования одинаковых идентификаторов в одной программе

10. Общая форма задания пространства имен имеет следующий вид.

====

#namespace <имя>{ . . . }

++++

11. Каким оператором обеспечивается доступ к пространстве имен namespace?

====

#::

====

12. Определите, что будет напечатано на экране в результате компиляции:

```
#include <iostream>
using namespace std;
namespace A {void fun(int i) { cout <<i-2;} }
namespace B {void fun(int j) { cout <<j+2;} }
int main() {int i=12;  B::fun(i);  return 0;}
```

====

#14

====

13. Определите, что будет напечатано на экране в результате компиляции:

```
#include <iostream>
using namespace std;
namespace A {void fun(int i) { cout <<i+2;} }
namespace A {void fun(float i) { cout <<i-2;} }
int main() { float i=12;   A::fun(i);  return 0; }
```

#10

====

14. Определите, что будет напечатано на экране в результате компиляции:

```
#include <iostream>
using namespace std;
namespace A {  void fun(int i) { cout <<i+2;}
  namespace B {    void fun(int i) { cout <<i-2;}   }}
using namespace A;
int main() {int  i=12;  fun(i); return 0;}
```

====

#14

++++

15. Определите, что будет напечатано на экране в результате компиляции:

```
#include <iostream>
using namespace std;
namespace A {  void fun(int i) { cout <<i+2;}
  namespace B { void fun(int i) { cout <<i-2;} }}
using namespace A;
int main() {int  i=12;  B::fun(i); return 0;}
```

====

#10

++++

16. Определите, что будет напечатано на экране в результате компиляции:

```
#include <iostream>
using namespace std;
```

```
namespace A {  void fun(int i) { cout <<i+2;}
  namespace B {    void fun(int i) { cout <<i-2;}   }
namespace C=A::B;
int main() {int  i=12;  C::fun(i);  return 0; }
```

====

#10

++++

17. Какой оператор трансформации используется для приведение типов полиформных (Polimorf) видов?

====

#dynamic_cast

====

++++

18. Укажите на какой строке правильно написано синтаксис оператора static_cast:

====

#static_cast<тип имени>(значение)

====

19. Укажите на какой строке правильно написано синтаксис оператора dynamic _cast:

#dynamic_cast< тип *>(obyekt указатель) или dynamic_cast< тип &>(obyekt переменный)

====

20. Какой из операторов явного преобразования типов используется для изменения статистических типов данных?

====

#static_cast

====

++++

21. Библиотека контейнеров – это …

====

#представляет собой универсальный набор шаблонов классов и алгоритмов, которые позволяют программистам легко реализовывать структуры данных.

====

22. Контейнер – это …

====

#управляет памятью, выделенной для ее элементов, и предоставляет функции для доступа к ним напрямую или через итераторы.

====

++++

23. Найти правильный ответ для основных категорий контейнерных классов?

====

#Последовательный, ассоциативный (отсортированный и неупорядоченный) и адаптер

====

++++

24. Найти ответ, в котором последовательно указаны типы контейнеров?

====

#vector, array, deque, forward_list, list

25. Укажите класс шаблона array:

====

#template<class T, std::size_t N> struct array;

++++

26. Функция для удаления любого элемента из контейнера?

====

#erase()

====

27. Функция удаления последнего элемента из контейнера вектор?

====

#pop_back()

++++

28. Функция которая предоставляет доступ к указанному элементу с проверкой границ в контейнере array

====

#at

====

29. Функция которая предоставляет доступ к указанному элементу с проверкой границ в контейнере vector

====

#at

====

30. Функция которая предоставляет доступ к указанному элементу с проверкой границ в контейнере deque

====

#at

++++

31. Функция которая предоставляет доступ к указанный элемент (без проверки границ) в контейнере deque

====

#[]

====

++++

32. Функция которая предоставляет доступ к указанный элемент (без проверки границ) в контейнере array

#[]

====

++++

33. Функция которая предоставляет доступ к указанный элемент (без проверки границ) в контейнере vector

====

#[]

====

34. Какая функция предоставляет доступ к первому элементу в контейнере array

====

#front

====

35. Какая функция предоставляет доступ к первому элементу в контейнере vector

#front

====

36. Какая функция предоставляет доступ к первому элементу одно связанного списка (в контейнере forward_list)

====

#front

37. **Какая функция предоставляет доступ к первому элементу двух связанного списка (в контейнере list)**

====

#front

====

38. **Какая функция предоставляет доступ к первому элементу двухстороннего очереди (в контейнере deque)**

====

#front

====

39. **Какая функция предоставляет доступ к последнему элементу в контейнере array**

====

#back

++++

40. **Какая функция предоставляет доступ к к последнему элементу в контейнере vector**

====

#back

++++

41. **Какая функция предоставляет доступ к к последнему элементу одно связанного списка (в контейнере forward_list)**

====

#back

++++

42. **Какая функция предоставляет доступ к к последнему элементу двух связанного списка (в контейнере list)**

====

#back

++++

43. Какая функция предоставляет доступ к к последнему элементу двухстороннего очереди (в контейнере deque)

====

#back

++++

44. Какая функция в контейнере array проверяет, пустой ли контейнер

====

#empty

45. Какая функция в контейнере vector проверяет, пустой ли контейнер

====

#empty

====

46. Какая функция в контейнере forward_list проверяет, пустой ли контейнер

====

#empty

====

++++

47. Какая функция в контейнере list проверяет, пустой ли контейнер

====

#empty

++++

48. Какая функция в контейнере deque проверяет, пустой ли контейнер

====

#empty

====

49. Какая функция в контейнере array возвращает количество элементов?

====

#size

====

50. Какая функция в контейнере vector возвращает количество элементов?

====

#size

====

51. Какая функция в контейнере forward_list возвращает количество элементов?

====

#size

52. Какая функция в контейнере list возвращает количество элементов?

====

#size

====

53. Какая функция в контейнере deque возвращает количество элементов?

====

#size

====

54. Какая функция в контейнере forward_list вставляет элементы в начало списка?

====

#push_front

====

55. Какая функция в контейнере list вставляет элементы в начало списка?

====

#push_front

====

56. Какая функция в контейнере vector вставляет элементы в конец списка?

====

#push_back

++++

57. Какая функция в контейнере list вставляет элементы в конец списка?

====

#push_back

====

58. Какая функция в контейнере deque удаляет первый элемент контейнера

#pop_front

====

59. Какая функция в контейнере list удаляет первый элемент контейнера

====

#pop_front

====

++++

60. Какая функция в контейнере vector удаляет последний элемент массива

====

#pop_back

++++

61. Ассоциативные контейнеры …

====

#реализуют упорядоченные структуры данных с возможностью быстрого поиска.

====

++++

62. Укажите строку , где перечислены название упорядоченных ассоциативных контейнеров

====

#set, map, multiset, multimap

_set, multiset, unordered_multiset

++++

63. Контейнер set - это…

====

#набор уникальных ключей, отсортированных по ключу

++++

64. Контейнер map - это…

====

#коллекция пар ключ-значение, отсортированная по ключам, ключи являются уникальными

====

++++

65. Контейнер multiset - это…

====

#коллекция ключей, отсортированная по ключам

====

++++

66. multimap контейнер - это…

====

#набор пар ключ-значение, отсортированных по ключам, ключи не являются уникальными

====

67. Который из указанных функций, возвращает итератор, который указывает на первый элемент в наборе с ключом, большим или равным указанному ключу?

====

#lower_bound()

+++

68. Что является основой для сортировки в контейнере map?

====

#ключ (key)

====

69. Какой из методов указанных ниже, возвращает количество элементов в наборе, соответствующее ключу, заданному параметром, это…

====

#count(value)

====

++++

70. Найдите разницу между контейнером set и multiset.

====

#В контейнере set ключи считаются уникальными, а в multiset ключи могут дублироваться.

====

++++

71. Что будет напечатано на экране после выполнения фрагмента программы

```
set <int> st;
for(int i=1; i<10;i++;)
st.insert(i%4);
for (auto it: st)
cout<<st<<" ";
```

====

#0 1 2 3

====

++++

72. Что будет напечатано на экране после выполнения фрагмента программы

```
multiset <int> st;
for(int i=1; i<10;i++;)
  st.insert(i%4);
for (auto it: st)
 cout<<st<<" ";
```

#0 0 1 1 1 2 2 3 3

====

73. Что будет напечатано на экране после выполнения фрагмента программы

```
multimap <int, char> st;
```

```
pair <int, char> zap;
int i; char s;
    for (i=0; i<10;i++)
    {
        zap.first=i%4;
        zap.second='A'+i;
        st.insert(zap);
    }
    cout<<"\nst[1]="<<st[1];
```

====

#ошибка в компиляции

++++

74. Что будет напечатано на экране после выполнения фрагмента программы

```
multimap <int, char> st;
pair <int, char> zap;
int i; char s;
    for (i=0; i<10;i++)
    {
        zap.first=i%4;
        zap.second='A'+i;
        st.insert(zap);
    }
    cout<<"\nst[0]="<<st.begin()->first;
```

====

#st[0]=0

++++

75. Что будет напечатано на экране после выполнения фрагмента программы

```
multimap <int, char> st;
pair <int, char> zap;
int i; char s;
    for (i=0; i<10;i++)
    {
        zap.first=i%4;
        zap.second='A'+i;
        st.insert(zap);
    }
    for(auto it=st.begin(); it!=st.end(); it++)
        cout<<it->first<<" ";
```

====

0 0 0 1 1 1 2 2 3 3

====

++++

76. Что будет напечатано на экране после выполнения фрагмента программы

```
multimap <int, char> st;
pair <int, char> zap;
int i; char s;
    for (i=0; i<10;i++)
    {
        zap.first=i%4;
        zap.second='A'+i;
        st.insert(zap);
    }
```

```
    for(auto it=st.begin(); it!=st.end(); it++)
        cout<<it->second<<" ";
```

====

#A E I B F J C G D H

====

++++

77. Что будет напечатано на экране после выполнения фрагмента программы

```
map <int, char> st;
pair <int, char> zap;
int i; char s;
    for (i=0; i<10;i++)
    {
        zap.first=i;
        zap.second='A'+i;
        st.insert(zap);
    }
    for(auto it=st.begin(); it!=st.end(); it++)
        cout<<it->second<<" ";
```

====

#A B C D E F G H I J

++++

78. Что будет напечатано на экране после выполнения фрагмента программы

```
map <int, char> st;
pair <int, char> zap;
int i; char s;
    for (i=9; i>=0;i--)
```

```
        {
                zap.first=i%4;
                zap.second='A'+i;
                st.insert(zap);
        }
        cout<<"\nst[1]="<<st[1];
```

====

#st[1]=J

++++

79. Что будет напечатано на экране после выполнения фрагмента программы

```
multimap <int, char> st;
   pair <int, char> zap;
        multimap <int, char> :: iterator it,itb,ite;
   int i; char s;
        for (i=9; i>=0;i--)
        {
                zap.first=i%4;
                zap.second='A'+i;
                st.insert(zap);
        }
        itb=st.lower_bound(1);
   ite=st.upper_bound(1);
        for (it=itb; it!=ite; it++)
                cout<<it->second<<" ";
```

====

#J F B

====

++++

80. Что будет напечатано на экране после выполнения фрагмента программы

```
multimap <int, char> st;
   pair <int, char> zap;
      multimap <int, char> :: iterator it,itb,ite;
   int i; char s;
      for (i=0; i<10;i++)
      {
            zap.first=i%4;
            zap.second='A'+i;
            st.insert(zap);
      }
      itb=st.lower_bound(1);
   ite=st.upper_bound(1);
      for (it=itb; it!=ite; it++)
            cout<<it->second<<" ";
```

====

#B F J

====

81. В контейнере set функция count возвращает количество элементов в множестве, соответствующих данному ключу. Его значение

====

#0 или 1

====

82. В контейнере multiset функция count возвращает количество элементов в множестве, соответствующих данному ключу. Его значение

====

#[0,N] - N здесь количество элементов

====

++++

83. В ассоциативных контейнерах функция возвращающий итератор, указывающий на первый элемент, ключи которого соответствуют ключу, заданному параметром

====

#find

====

++++

84. Функция (в ассоциативных контейнерах) возвращает диапазон элементов, соответствующих определённому ключу - это ...

====

#egual_range

====

++++

85. Который из указанных функций, возвращает итератор на первый элемент больший, чем заданный ключ?

====

#upper_bound()

++++

86. Назначение функции max_size() в контейнере multimap?

====

#возвращает максимально возможное количество элементов

====

++++

87. Стеком называется ... данных построенной по принципу - последний элемент обслуживается первым. Найдите пропущенное слово?

====

#структура

++++

88. По какому принципу обслуживается данные из стека (stack)?

====

#LIFO

++++

89. Какой ответ правильно описывает значение слова «стек»?

=====

#Куча

====

++++

90. Какая функция удаляет верхний элемент стека?

====

#pop()

++++

91. По какому принципу обслуживается данные из очереди (queue)?

====

#FIFO

++++

92. В какой строке правильно указано расшифровка аббревиатуры FIFO ?

====

#first in first out

++++

93. В какой строке указано адаптер "двухсторонний очередь" ?

====

#deque

++++

94. Укажите строку, в которой правильно указана функция добавления элемента в конец двусторонней очереди?

====

#push.back()

++++

95. Что выйдет на экран после выполнения фрагмента программы?

```
stack<int> mystack;
  mystack.push(1);
      if (mystack.empty()) {       cout << "True";    }
  else {       cout << "False";   }
```

====

#False

++++

96. Что выйдет на экран после выполнения фрагмента программы?

```
int sum = 0;
```

stack<int> mystack;

mystack.push(1); mystack.push(8); mystack.push(3);

mystack.push(6); mystack.push(2);

while (!mystack.empty()) {

sum = sum + mystack.top(); mystack.pop(); }

cout << sum;

====

#20

++++

97. Что выйдет на экран после выполнения фрагмента программы?

int sum = 0;

stack<int> mystack;

mystack.push(1); mystack.push(8); mystack.push(3);

mystack.push(6); mystack.push(2);

cout << mystack.size();

====

#5

====

++++

98. Что выйдет на экран после выполнения фрагмента программы?

queue<int> myqueue;

myqueue.push(0);

myqueue.push(1);

myqueue.push(2);

while (!myqueue.empty()) {

cout << ' ' << myqueue.front();

myqueue.pop(); }

====

#0 1 2

++++

99. Что выйдет на экран после выполнения фрагмента программы?

queue<int> myqueue;

myqueue.push(0); myqueue.push(1); myqueue.push(2);

myqueue.pop(); myqueue.pop();

while (!myqueue.empty()) { cout << ' ' << myqueue.front();

myqueue.pop(); }

====

#2

====

++++

100. Что выйдет на экран после выполнения фрагмента программы?

int c = 0;

queue<int> myqueue;

myqueue.push(5); myqueue.push(13); myqueue.push(0);

myqueue.push(9); myqueue.push(4);

while (!myqueue.empty()) { myqueue.pop(); c++; }

cout << c;

====

#5

====

++++

101. Что выйдет на экран после выполнения фрагмента программы?

```
queue<int> myqueue;
  myqueue.push(3);   myqueue.push(4);   myqueue.push(1);
myqueue.push(7);   cout << myqueue.front();
```

====

#3

++++

102. Что выйдет на экран после выполнения фрагмента программы?

```
queue<int> myqueue;
  myqueue.push(8);  myqueue.push(7);   myqueue.push(6);
  myqueue.push(5);   myqueue.push(4);   myqueue.push(3);
  myqueue.push(2);   myqueue.push(1);
  if (myqueue.front() > myqueue.back()) {
     cout << myqueue.front() - myqueue.back();   }
  else if (myqueue.front() < myqueue.back()) {
     cout << myqueue.back() - myqueue.front();   }
  else      cout << "0";
```

====

#7

++++

103. Что выйдет на экран после выполнения фрагмента программы?

```
queue<int> myqueue;
  myqueue.push(8);   myqueue.push(7);   myqueue.push(6);
  myqueue.push(5);   myqueue.push(4);   myqueue.push(3);
```

myqueue.push(2); myqueue.push(1);

if (myqueue.front() < myqueue.back()) {

cout << myqueue.front() - myqueue.back(); }

else if (myqueue.front() == myqueue.back()) {

cout << myqueue.back() - myqueue.front(); }

else cout << "0";

====

#0

====

++++

104. Что означает термин "функтор", дайте определение.

====

#Функтор — это сокращение от функциональный объект, представляющий собой конструкцию, позволяющую использовать объект класса как функцию

"

====

#класс, который реализует operator(), называется функтором.

====

++++

105. Как называется коллекция данных (контейнер), которая построена по принципу LILO (last in — last out: последним пришел — последним вышел)?

====

#queue

====

++++

106. Как называется коллекция данных (контейнер), которая построена по принципу LIFO (Last In — First Out)?

====

#stack

++++

107. Как называется коллекция данных (контейнер), которая построена по принципу - кого больше веса тот обслуживается в первую очередь?

====

#priority_queue

++++

108. В каком адаптере определены только операции push_back, pop_back и pop_front?

====

#stack

++++

109. В каком из адаптеров элементы расположены в убывающем порядке?

====

#priority_queue

++++

110. Укажите строку, в которой правильно указана функция для удаленя элемента из конца двусторонней очереди?

====

#pop.back()

++++

111. Назначение функции all_of из библиотеки algorithm

====

#Проверяет, что предикат p возвращает значение true для всех элементов в диапазоне [first, last).

====

++++

112. Назначение функции any_of из библиотеки algorithm

====

#Проверяет, что предикат p возвращает значение true для хотя бы одного элемента в диапазоне [first, last).

++++

113. Назначение функции none_of из библиотеки algorithm

====

#Проверяет, что предикат p не возвращает значение true ни для одного элемента в диапазоне [first, last).

====

++++

114. Назначение функции for_each из библиотеки algorithm

====

====

#По порядку применяет заданный функциональный объект f к результату разыменования каждого итератора в диапазоне [first, last).

++++

115. Назначение функции count(InputIt first, InputIt last, const T &value) из библиотеки algorithm

====

#Возвращает количество элементов в диапазоне [first, last) равные value.

====

++++

116. Назначение функции count_if(InputIt first, InputIt last, UnaryPredicate p) из библиотеки algorithm

====

#Возвращает количество элементов в диапазоне [first, last), для которых предикат p возвращает значение true.

====

++++

117. Назначение функции mismatch(InputIt1 first1, InputIt1 last1, InputIt2 first2) из библиотеки algorithm

#Возвращает первую пару несовпадающих элементов из двух диапазонов: одного, определяемого [first1, last1), и другого, начинающегося с first2.

====

++++

118. Назначение функции find_first_of(InputIt first, InputIt last,ForwardIt s_first, ForwardIt s_last) из библиотеки algorithm

====

#Ищет в диапазоне [first, last) любой элемент диапазона [s_first, s_last).

++++

119. Назначение функции find(InputIt first, InputIt last, const T& value) из библиотеки algorithm

====

#функции находят в диапазоне [first, last) первый элемент, равный value.

++++

120. Назначение функции find_end(ForwardIt1 first, ForwardIt1 last,ForwardIt2 s_first, ForwardIt2 s_last) из библиотеки algorithm

====

#Ищет последнее вхождение подпоследовательности элементов [s_first, s_last) в диапазон [first, last).

++++

121. Назначение функции find_if(InputIt first, InputIt last, UnaryPredicate p) из библиотеки algorithm

====

#функции находят в диапазоне [first, last) первый элемент, для которого предикат p возвращает значение true.

====

122. Назначение функции find_if_not(InputIt first, InputIt last,UnaryPredicate q) из библиотеки algorithm

====

#функции находят в диапазоне [first, last) первый элемент, для которого предикат q возвращает значение false.

====

++++

123. Назначение функции adjacent_find(ForwardIt first, ForwardIt last) из библиотеки algorithm

====

#Ищет в диапазоне [first, last) два одинаковых смежных элемента.

====

++++

124. Назначение функции copy(InputIt first, InputIt last, OutputIt d_first) из библиотеки algorithm

====

#Копирует элементы диапазона [first, last) в диапазон, начинающийся с d_first.

====

++++

125. Назначение функции copy_if(InputIt first, InputIt last, OutputIt d_first,UnaryPredicate pred) из библиотеки algorithm

====

#Копирует только те элементы, для которых предикат pred возвращает true.

====

126. Назначение функции copy_backward(BidirectionalIterator1 first,BidirectionalIterator1 last, BidirectionalIterator2 d_last) из библиотеки algorithm

====

#Копирует элементы из промежутка [first, last) в промежуток, кончающийся в d_last. Элементы копируются в обратном порядке (последний элемент копируется первым), но их относительный порядок сохраняется.

++++

127. Назначение функции move(InputIt first, InputIt last, OutputIt d_first)из библиотеки algorithm

====

#Перемещает элементы из диапазона [first, last) в другой диапазон, начинающийся с d_first.

========++++

128. Назначение функции move_backward(BidirIt1 first, BidirIt1 last, BidirIt2 d_last)из библиотеки algorithm

====

#Перемещает диапазон элементов в новое место в обратном порядке

++++

129. Назначение функции fill(ForwardIt first, ForwardIt last, const T& value) из библиотеки algorithm

#Присваивает диапазону элементов определённое значение

130. Назначение функции fill_n(OutputIt first, Size count, const T& value)из библиотеки algorithm

====

#Если count>0, присваивает заданное значение value первым count элементам в диапазоне, начинающемся с first. Иначе ничего не делает.

++++

131. Назначение функции generate(ForwardIt first, ForwardIt last, Generator g) из библиотеки algorithm

====

#Присваивает каждому элементу диапазона [first, last) значение, сгенерированное заданным функциональным объектом g.

====

132. Назначение функции generate_n(OutputIt first, Size count, Generator g) из библиотеки algorithm

====

#Если count>0, присваивает значения, сгенерированные заданным функциональным объектом g, первым count элементам диапазона, начинающегося с first. Иначе ничего не делает.

====

133. Назначение функции remove(ForwardIt first, ForwardIt last, const T& value) из библиотеки algorithm

====

#Удаляет из диапазона [first, last) все элементы, равные value

====

134. Назначение функции remove_if(ForwardIt first, ForwardIt last, UnaryPredicate p) из библиотеки algorithm

====

#Удаляет все элементы, для которых предикат p возвращает true.

++++

135. Назначение функции remove_copy(InputIt first, InputIt last, OutputIt d_first, const T& value) из библиотеки algorithm

====

#Копирует элементы из диапазона [first, last) в диапазон, начинающийся с d_first, кроме тех элементов,

значение которых равен value

====

++++

136. Назначение функции remove_copy_if(InputIt first, InputIt last, OutputIt d_first, UnaryPredicate p) из библиотеки algorithm

====

#Копирует элементы из диапазона [first, last) в диапазон, начинающийся с d_first, кроме тех элементов, для которых предикат p возвращает true

+++

137. Назначение функции replace(ForwardIt first, ForwardIt last,const T& old_value, const T& new_value) из библиотеки algorithm

====

#Заменяет все элементы в диапазоне [first, last),равные old_value, на new_value.

====

138. Назначение функции replace_if(ForwardIt first, ForwardIt last,UnaryPredicate p, const T& new_value) из библиотеки algorithm

====

#Заменяет все элементы в диапазоне [first, last), для которых предикат p возвращает true, на new_value.

++++

139. Назначение функции swap(T& a, T& b)из библиотеки algorithm

====

#Меняет местами значения a и b.

====

++++

140. Назначение функции swap(T2 (&a)[N], T2 (&b)[N])из библиотеки algorithm

====

#Обмен массивов a и b.

====

++++

141. Назначение функции swap_ranges(ForwardIt1 first1, ForwardIt1 last1, ForwardIt2 first2)из библиотеки algorithm

#Обмен элементов между диапазоном [first1, last1) и другим диапазоном, который начинается с first2. Количество элементов в этих двух диапазонах должно совпадать.

====

++++

142. Назначение функции iter_swap(ForwardIt1 a, ForwardIt2 b)из библиотеки algorithm

====

#Меняет местами значения элементов, на которые указывают два тератора.

++++

143. Назначение функции reverse(BidirIt first, BidirIt last);из библиотеки algorithm

====

#Меняет порядок следования элементов в диапазоне [first, last) на противоположный.

++++

144. Назначение функции rotate(ForwardIt first, ForwardIt n_first, ForwardIt last)

из библиотеки algorithm

====

#Меняет местами элементы в диапазоне [first, last) таким образом,

что элемент n_first становится первым в новом диапазоне, а n_first-1 — последним.

====

++++

145. Назначение функции unique(ForwardIt first, ForwardIt last)
из библиотеки algorithm

#Удаляет все последовательно повторяющиеся элементы из диапазона [first, last) и возвращает итератор на элемент, следующий за последним элементом нового диапазона.

++++

146. Назначение функции merge(InputIt1 first1, InputIt1 last1,InputIt2 first2, InputIt2 last2, OutputIt d_first) из библиотеки algorithm

====

#Объединяет два отсортированных диапазона [first1, last1) и [first2, last2), записывая элементы в новый диапазон начиная с d_first.

++++++++++++++++++

147. Назначение функции set_difference из библиотеки algorithm

====

#вычисляет разницу между двумя наборами

====

++++

148. Назначение функции set_intersection из библиотеки algorithm

====

#вычисляет пересечение двух множеств

++++

149. Назначение функции set_symmetric_difference из библиотеки algorithm

====

#вычисляет симметричную разницу между двумя наборами

====

++++

150. Назначение функции set_union из библиотеки algorithm====

#вычисляет объединение двух множеств

++++

151. Назначение функции max_element из библиотеки algorithm

====

#возвращает наибольший элемент в диапазоне

====

152. Назначение функции min_element из библиотеки algorithm

====

#возвращает наименьший элемент в диапазоне

====

++++

153. Назначение функции minmax_element из библиотеки algorithm

====

#возвращает наименьший и наибольший элементы в диапазоне

====

++++

154. Назначение функции equal из библиотеки algorithm

====

#определяет, одинаковы ли два множества элементов

155. Назначение функции accumulate из библиотеки algorithm

====

#суммирует диапазон элементов

====

++++

156. Назначение функции inner_product из библиотеки algorithm

====

#вычисляет скалярное произведение двух диапазонов элементов

====

++++

157. Назначение функции adjacent_difference из библиотеки algorithm

====

#вычисляет различия между соседними элементами в диапазоне

====

++++

158. Назначение функции partial_sum из библиотеки algorithm

====

#вычисляет частичную сумму диапазона элементов

++++

159. Какая функция выполняет поиск первого вхождения заданного значения в контейнере?

====

#find()

++++

160. Что обеспечивают доступ к элементам контейнера?

====

#iterator

====

++++

161. Какая функция возвращает итератор, который указывает на первый элемент контейнера (при наличии в контейнере элементов)?

====

#begin()

====

++++

162. Какая функция возвращает итератор, который указывает на следующую позицию после последнего элемента, то есть по сути на конец контейнера?

====

#end()

====

++++

163. Какое выражение дает возможность получение значение элемента, на который указывает итератор?

====

#*iter

====

++++

164. Какое выражение дает возможность перемещение итератора вперед для обращения к следующему элементу?

====

#++iter

+++++

165. Какое выражение дает возможность перемещение итератора назад для обращения к предыдущему элементу?

====

#--iter

====

++++

166. Какая операция возвращает итератор, который смещен от итератора iter на n позиций вперед?

====

#iter + n

====

++++

167. Какая операция возвращает итератор, который смещен от итератора iter на n позиций назад?

====

#iter – n

====

168. Какая операция перемещает итератор на n позиций вперед?

====

#iter += n

====

++++

169. Какая операция перемещает итератор на n позиций назад?

====

#iter –= n

====

170. Какая операция возвращает количество позиций между итераторами iter1 и iter2?

====

#iter1 – iter2

====

171. Какая функция принимает вспомогательную функцию в качестве третьего параметра, что позволяет выполнять сортировку так, как нам это захочется?

====

#sort()

====

++++

172. Что выйдет на экран после выполнения фрагмента программы

```
valarray<int> sample(12);
for (int i = 0; i < 13; ++i)  sample[i] = i;
valarray<int> bar = sample[std::slice(2, 3, 4)];
cout << "slice(2, 3, 4):";
for (size_t n = 0; n < bar.size(); n++) cout << ' ' << bar[n];
```

=====

#slice (2, 3, 4): 2 6 10

====

173. Что выйдет на экран после выполнения фрагмента программы

```
valarray<int> sample (14);
for (int i=0; i<14; ++i) sample[i]=i;
size_t start=1; size_t lengths[]= {2,3};  size_t strides[]= {7,2};
gslice mygslice (start,
valarray<size_t>(lengths,2),valarray<size_t>(strides,2));
valarray<int> data = sample[mygslice];
cout << "gslice:";
for (int i=0; i<data.size(); i++)  cout << ' ' << data[i];
```

====

#gslice: 1 3 5 8 10 12

====

++++

174. Что выйдет на экран после выполнения фрагмента программы

```
valarray<int> varr = { 20, 40, 60, 80 };
cout << "The size of valarray is: ";
cout << varr.size()<< endl;
```

=====

#The size of valarray is: 4

====

++++

175. Что выйдет на экран после выполнения фрагмента программы

```
valarray<int>  varr = { -20, 40, -50, 60, 80, 0, 0 };
cout << "The size of valarray is: ";
cout << varr.size()<< endl;
```

====

#The size of valarray is: 7

176. Что выйдет на экран после выполнения фрагмента программы

```
alarray<int> varr = { 10, 20, 30, 40, 50 };
cout << "valarray contains=";
for (auto i = begin(varr); i != end(varr); i++) {
      cout << ' ' << *i;     }
```

====

#valarray contains = 10 20 30 40 50

====

177. Что выйдет на экран после выполнения фрагмента программы

```
valarray<int> varr = { -10, -20, -30, -40 };
cout << "valarray contains=";
for (auto i = begin(varr); i != end(varr); i++) {
      cout << ' ' << *i;     }
```

====

#valarray contains = -10 -20 -30 -40

++++

178. Что выйдет на экран после выполнения фрагмента программы

```
valarray<int> varr = { 15, 10, 30, 33, 40 };
    cout << "The sum of valarray is =  "  << varr.sum() << endl;
```

====

#The sum of valarray is = 128

====

++++

179. Что выйдет на экран после выполнения фрагмента фрагмента программы

```
valarray<int> varr = { 1, 2, 3, 4, 5 };
  cout << "The sum of valarray is = "  << varr.sum() << endl;
```

====

#The sum of valarray is =15

+++++

180. Что выйдет на экран после выполнения фрагмента программы

```
complex<double> mycomplex(10.0, 2.0);
cout << "Real part: " << real(mycomplex) << endl;
out << "Imaginary part: " << imag(mycomplex) << endl;
```

====

#Real part: 10

Imaginary part: 2

====

++++

181. Что выйдет на экран после выполнения фрагмента программы

```
typedef complex<double> point;
```

```
#define x real()
#define y imag()
 int main()  {     point P(2.0, 3.0);
   cout << "The X-coordinate of point P is: " << P.x << endl;
   cout << "The Y-coordinate of point P is: " << P.y << endl;
    return 0; }
```

====

#The X-coordinate of point P is: 2

The Y-coordinate of point P is: 3

====

++++

182. Что выйдет на экран после выполнения фрагмента программы

```
cout << "Square root of -9 is =";
out << sqrt(complex<double>(-9, 0)) << endl;
cout << "Square root of (-9, -0), is = ";
cout << sqrt(complex<double>(-9, -0.0)) << endl;
```

=====

#Square root of -9 is =(0, 3)

====

+++++

183. Что выйдет на экран после выполнения фрагмента программы

```
valarray<int> varr = { 3, 2, 1, 4, 5 };
cout << "The smallest element" << " of valarray is = "
       << varr.min() << endl;
```

====

#The smallest element of valarray is = 1

++++

184. Что выйдет на экран после выполнения фрагмента программы

valarray<int> varr = { 22, 24, 36, 42, 12 };
cout << "The smallest element" << " of valarray is = "
<< varr.min() << endl;

====

#The smallest element of valarray is = 12

====++++

185. Какая функция применяет манипуляции, указанные в ее аргументах, ко всем элементам valarray одновременно и возвращает новый valarray с манипулируемыми значениями?

=====

#apply ()

====

186. Какая функция возвращает суммирование всех элементов valarray одновременно?

====

#sum ()

====

++++

187. Какая функция задает подмножество, диапазон значений массива, которые будут обработаны тем или иным образом?

====

#slice()

+++++

188. Что означает первый параметр функции slice()?

====

#индекс элемента массива, с которого будет начинаться выборка, если индекс равен 0, значит выборка начнется с первого элемента массива

====

189. Что означает второй параметр функции slice()?

====

#количество элементов в массиве, которые подлежат выборке

====

++++

190. Что означает третий параметр функции slice()?

====

#шаг выборки, если шаг равен 2, а значит каждый второй элемент попадет в подмножество

191. Какой специальный контейнер, который появился в C ++ 98 и используется для эффективного хранения и обеспечения математических операций над массивами?
====

#valarray

192. Когда вы нажимаете Ctrl + Alt + L в режиме приложения Windows Forms в Visual Studio ...

====

#Откроется окно Solution Exploler там, где последний раз открывался, в главном диалоговом окне.

====

++++

193. Чтобы открыть окно Solution Exploler в режиме приложения Windows Forms в Visual Studio ...

====

#View-> Solution Exploler

====

++++

194. Чтобы открыть окно панели инструментов компонентов Visual Studio в режиме приложения Windows Forms ...

====

#View-> Toolbox

====

++++

195. Когда вы нажимаете Ctrl + Alt + X в режиме приложения Windows Forms в Visual Studio...

====

#Откроется окно Component Toolbox там, где последний раз открывался, в главном диалоговом окне.

====

++++

196. Когда вы нажимаете Ctrl + \E в режиме приложения Windows Forms в Visual Studio...

====

#В нижней части главного диалогового окна откроется окно ошибок компиляции (Error List)

====

++++

197. Чтобы просмотреть ошибки, обнаруженные в процессе компиляции…

====

#View-> Error List

====

++++

198. Чтобы управлять (закрывать, добавлять) панель инструментов главного диалогового окна, нужно ...

====

#View-> Toolbars

====

++++

199. Чтобы скомпилировать проект, нужно ...

====

#Ctrl+Shift+B или Build -> Build Solution

====

++++

200. Чтобы запустить проект на выполнения, нужно ...

====

#F5 или Debug -> Start Debugging

====

++++

201. Чтобы добавить в проект новый класс, нужно ...

====

#Project -> Add Class

====

++++

202. **Чтобы добавить в проект файл формата cpp, нужно** ...

====

#Project -> Add New Item

====

++++

203. Чтобы добавить в проект файл формата cpp, нужно ...

====

#Ctrl + Shift + A

====

++++

204. Для чего предназначен компонент Label?

====

#Компонент Label предназначен для отображения текстовой информации.

====

++++

205. Для чего предназначен компонент TextBox?

====

#Компонент TextBox предназначен для ввода данных с клавиатуры.

====

++++

206. Для чего предназначен компонент Button?

====

Компонент предназначен для выполнения команд.

====