1. Что определяет класс? Чем отличается класс от объекта?  
Класс определяет структуру данных и функции над ними. Объект - это экземпляр класса.  
  
2. Можно ли объявлять массив объектов? А массив классов?  
Да, можно.  
Например:  
class A {};  
A a[4];  
Массив классов определить только используя мета-программирование, например boost::mpl::vector, boost::tuple.  
  
3. Разрешается ли объявлять указатель на объект? А указатель на класс?  
Да, нет. Это звёздочка перед переменной  
  
4. Допускается ли передавать объекты в качестве параметров, и какими способами? А возвращать как результат?  
По ссылке, по значению, по указателю. Так звучит классический ответ. На самом деле только по значению. Когда передается указатель, то на стеке заводится переменная размером с указатель, с типом "указатель". И он передается всё-таки по значению. Указатели и ссылки - это разные ипостаси так называемого ссылочного типа. Отличие ссылки от указателя в том, что она более безопасна. Так, мы не можем присвоить ссылке значение null, как мы это можем сделать с указателем. Также мы не можем переопределить ссылку и заставить её ссылаться на другую переменную.  
Как результат можно возвращать значение, ссылку или указатель.  
  
5. Как называется использование объекта одного класса в качестве поля другого класса?  
Композиция.  
  
6. Является ли структура классом? Чем класс отличается от структуры?  
Да, является. Класс отличается от структуры тем, что у него по умолчанию поля объявляются как private. Наследование у класса тоже private. У структуры поля по умолчанию - public, наследование тоже public.  
Еще в объявлении шаблона нельзя писать struct, т.е. это не скомпилируется:  
template class C{}; // ошибка  
  
7. Какие ключевые слова в C++ обозначают класс?  
class, struct, union (особый вид класса, который не может быть унаследован и не может быть базовым классом для других, также не может иметь виртуальных функций).  
  
8. Объясните принцип инкапсуляции.  
Сокрытие данных класса, отделение интерфейса от реализации.  
  
9. Что такое композиция?  
Аггрегация - отношение часть-целое, когда один класс является частью другого.  
Композиция - это аггрегация, с временем жизни аггрегируемого объекта, равным времени жизни аггрегатора.  
  
10. Для чего используются ключевые слова public и private?  
Для задания области видимости полей и методов.  
  
11. Можно ли использовать ключевые слова public и private в структуре?  
Да.  
  
12. Существуют ли ограничения на использование public и private в классе? А в структуре?  
Нет.  
  
13. Обязательно ли делать поля класса приватными?  
Нет.  
  
14. Что такое метод? Как вызывается метод?  
Метод - это функция-член класса. Может вызываться с помощью оператора "->", если мы имеем указатель на объект. Либо с помощью оператора ".".  
  
15. Может ли метод быть приватным?  
Да.  
  
16. Как определить метод непосредственно внутри класса? А вне класса? Чем эти определения отличаются?  
class A  
{  
  void f() {std::cout << "f()" << std::endl;}  
};  
class A  
{  
  void f();  
};  
void A::f() { std::cout << "f()" << std::endl;}  
Эти определения отличаются лишь местом, где они находятся.  
  
17. Можно в методах присваивать параметрам значения по умолчанию?  
Да.  
  
18. Что обозначается ключевым словом this?  
this - это неявный параметр, который передается во все методы класса (кроме статических и friend). this - это указатель на текущий объект, экземпляр класса. Если у нас имеется класс A, то в const методах this имеет тип const A\*, в неконстантных - A\*, для volatile методов - volatile A\*, для const volatile методов - const volatile A\*.  
  
19. Зачем нужны константные методы? Чем отличается определение константного метода от обычного?  
Метод, объявленный как константный - гарантирует, что состояние объекта не будет измененено. Константный метод вызывается для объекта, объявленного как const.  
  
20. Может ли константный метод вызываться для объектов-переменных? А обычный метод - для объектов-констант?  
Да. Да, если использовать const\_cast.  
  
21. Объясните принцип полиморфизма.  
Полиморфизм - возможность работать с объектами разных типов одинаковым образом. Бывает статический (на этапе компиляции, шаблоны) и динамический (на этапе выполнения).  
  
22. Сколько места в памяти занимает объект класса? Как это узнать?  
class A {}; sizeof(A);  
  
23. Каков размер пустого объекта?  
С++ 11, 9.4. Complete objects and member subobjects of class type shall have nonzero size. Base class subobjects are not so constrained.  
  
24. Влияют ли методы на размер объекта?  
Нет. Только виртуальные. Если есть хотя бы один виртуальный метод - то создается указатель на таблицу виртуальных функций.  
  
25. Одинаков ли размер класса и аналогичной структуры?  
Да.  
  
26. Какие операции нельзя перегружать? Как вы думаете, почему?  
. .\* :: ?: sizeof() typeid()  
Если бы можно было перегрузить оператор . , то возникла бы путаница)  
  
27. Можно ли перегружать операции для встроенных типов?  
Нет.  
  
28. Можно ли при перегрузке изменить приоритет операции?  
Нет.  
  
29. Можно ли определить новую операцию?  
Нет.  
  
30. Перечислите особенности перегрузки операций как методов класса. Чем отличается перегрузка внешним образом от перегрузки как метода класса?  
При перегрузке внешним образом  
\* Нет доступа к private и protected полям и методам. Только если оператор объявлен как friend.  
\* Нельзя определить версии const.  
\* Нужно писать дополнительный параметр в аргументах.  
При перегрузке как метода класса:  
\* Есть доступ к private и protected полям и методам.  
\* Можно определить const версии.  
\* Для бинарных операторов не нужно писать первый параметр, потому что он уже имеется в виде this указателя. Для унарных операторов вообще не нужно писать аргументов по той же причине.  
Бинарные операторы лучше перегружать внешним образом. Пример:  
A a(3);  
A b(2);  
a < b; // Будет работать в обоих случаях.  
a < 2; // Будет работать в обоих случаях.  
3 < a; // Не будет работать, если оператор "<" определен как метод класса.  
  
31. Какой результат должны возвращать операции с присваиванием?  
\*this. Тип - неконстантная ссылка.  
  
32. Как различаются перегруженная префиксная и постфиксная операции инкремента и декремента?  
Объявлением, поведением и возвращаемым значением.  
A& operator++ (int) // Prefix.  
{  
  (\*this).a++;  
  return \*this;  // Returns lvalue, thus can be chained.  
}  
A operator++ () // Postfix  
{  
  A x(\*this);  
  (\*this).a++;  
  return x; // Returns rvalue, thus can not be chained.  
}  
  
33. Что означает выражение \*this? В каких случаях оно используется?  
Это разыменование указателя на текущий объект. Используется, если нужно вернуть ссылку на объект.  
  
34. Какие операции не рекомендуется перегружать как методы класса?  
"<", ">", "==", "<=", ">=", "<<", ">>"  
  
35. Какие операции разрешается перегружать только как методы класса?  
Те, которые требуют lvalue в качестве параметра.  
= [] -> ()  
type cast  
  
36. Дайте определение дружественной функции. Как объявляется дружественная функция? А как определяется?  
Дружественная функция - это функция, имеющая доступ к private и protected областям класса.  
Объявляется с помощью квалификатора friend внутри класса, к данным и методам которого она должна иметь доступ. Определяется либо внутри, там где находится объявление, либо снаружи класса.  
class A  
{  
    int a;  
public:  
    friend void f(A& \_a);  
};  
void f(A& \_a) { \_a.a = 1;}  
  
37. Дайте определение конструктора. Каково назначение конструктора? Перечислите отличия конструктора от метода.  
C++ 11. 12.2. A constructor is used to initialize objects of its class type. Because constructors do not have names, they are never found during name lookup; however an explicit type conversion using the functional notation (5.2.3) will cause a constructor to be called to initialize an object. [ Note: For initialization of objects of class type see 12.6. —end note ]  
Конструктор - это специальная функция, которая используется для инициализации объекта класса. Имя её совпадает с именем класса. Если конструктор без параметров и конструктор копии явно не определены пользователем, компилятор генерирует версии по умолчанию. Конструктор вызывается автоматически, когда создается объект класса. У конструктора нет возвращаемого значения. Внутри конструктора нет доступа до таблицы виртуальных функций, т.к. она еще не создана. Конструктор не может быть static, const, virtual, volatile, const volatile. Метод - может и может возвращать значение.  
  
38. Сколько конструкторов может быть в классе? Допускается ли перегрузка конструкторов? Какие виды конструкторов создаются по умолчанию.  
Сколько угодно. Да, перегрузка допускается. Конструктор по умолчанию, конструктор копии.  
  
39. Может ли конструктор быть приватным? Какие последствия влечет за собой объявление конструктора приватным.  
Да. Невозможность создать объект класса на стеке.  
  
40. Приведите несколько случаев, когда конструктор вызывается неявно.  
Передача аргумента по значению.  
Приведение типа (опять же, когда передается аргументом в оператор или функцию).  
Возврат по значению (кроме того случая, когда результат присваивается константной ссылке на класс const A&, в этом случае конструктор не вызывается, а ссылка ссылается на временный объект).  
  
41. Как проинициализировать динамическую переменную?  
malloc  
new  
new (nothrow)  
placement new  
  
42. Как объявить константу в классе? Можно ли объявить дробную константу?  
Дробную константу объявить можно.  
class A  
{  
  const int a;  
  static const int b = 10;  
  static const int c;  
  static const double d;  
public:  
  A(): a(1) {};  
};  
const int A::c = 5;  
const double A::d = 10.;  
  
43. Каким образом разрешается инициализировать константные поля в классе?  
В списке инициализации. См. 42.  
  
44. В каком порядке инициализируются поля в классе? Совпадает ли этот порядок с порядком перечисления инициализаторов в списке инициализации конструктора?  
В порядке объявления внутри класса. Необязательно.  
  
45. Какие конструкции C++ разрешается использовать в списке инициализации в качестве инициализирующих выражений?  
?:  
rvalue  
инициализированные lvalue  
  
46. Какой вид конструктора фактически является конструктором преобразования типов?  
Не explicit конструктор с параметром.  
class A  
{  
  int a;  
public:  
  A(int \_b){ a = \_b;}  
};  
void foo(A \_a) { std::cout << "Casted" << std::endl; }  
  
47. Для чего нужны функции преобразования? Как объявить такую функцию в классе?  
Для преобразования типов.  
class B  
{  
    int b;  
public:  
    B(int \_b): b(\_b){}  
};  
class A  
{  
    int a;  
public:  
    A(int \_b){ a = \_b;}  
    operator B()  
    {  
      return B(a);  
    }  
};  
  
48. Как запретить неявное преобразование типа, выполняемое конструктором инициализации?  
Объявить конструктор инициализации как explicit.  
  
49. Какие проблемы могут возникнуть при обределении функций преобразования?  
Может возникнуть конфликт между оператором преобразования типа и конструктором инициализации.  
  
50. Для чего служит ключевое слово explicit?  
Это ключевое слово ставится перед конструктором. Служит для предотвращения неявного вызова конструктора.  
  
51. Влияет ли наличие целочисленных констант-полей на размер класса?  
Да, если эти поля не являются static.  
  
52. Разрешается ли объявлять массив в качестве поля класса? Как присвоить элементам массива начальные значения?  
Да. В теле конструктора. Если статический и константный - то вне класса. Также можно использовать std::array.  
  
53. Сколько операндов имеет операция индексирования []? Какой вид результата должна возвращать эта операция?  
Один. Какой угодно. Лучше ссылку, чтобы это было lvalue и ей можно было присваивать значения.  
  
54. Для чего нужны статические поля в классе? Как они определяются?  
Для того, чтобы иметь один экземпляр этого поля на все объекты. Определяются вне класса.  
  
55. Как объявить в классе и проинициализировать статический константный массив.  
class A  
{  
    static const int b[3];  
public:  
    A(){}  
};  
const int A::b[3] = {1,2,3};  
  
56. Что такое выравнивание и от чего оно зависит? Влияет ли выравнивание на размер класса?  
Выравнивание - это то, как компилятор располагает поля класса в памяти. Зависит от компилятора. Влияет на размер класса. Отключить выравнивание можно с помощью директив:  
#pragma pack(push) // или #pragma pack(push, 1)  
// объявление структуры или класса.  
#pragma pack(pop)  
ISO IEC 14882 2011: 3.11 ... . An alignment is an implementation-defined integer value representing the number of bytes between successive addresses at which a given object can be allocated. An object type imposes an alignment requirement on every object of that type; stricter alignment can be requested using the alignment specifier (7.6.2).  
  
57. Дайте определение контейнера.  
Контейнер - класс, предназначенный для хранения объектов.  
  
58. Какие виды встроенных контейнеров в C++ вы знаете?  
C++03: vector, list, deque, stack, queue, priority\_queue, map, set, multimap, multiset, bitset.  
C++11 (добавились): unordered\_map, unordered\_set, array, forward\_list.  
  
59. Какие виды доступа к элементам контейнера вам известны?  
Итеративный (путем разыменования итератора), прямой(по индексу), ассоциативный (по объекту).  
  
60. Чем отличается прямой доступ от ассоциативного?  
Прямой доступ - доступ по индексу, как в обычном массиве. Используется с контейнером vector.  
Ассоциативный - доступ по объекту. Используется с контейнерами - хешами, такими как map, multimap, set, multiset.  
  
61. Перечислите операции, которые обычно реализуются для последовательного доступа к элементам контейнера.  
begin(), end(), ++, --, \* (разыменование), -> (доступ по указателю), ==  
  
62. Дайте определение итератора.  
Итератор - это любой объект, который указывает на элемент из множества элементов (таких как массив или контейнер), умеет итерировать по элементом этого множества используя набор операторов (как минимум \* и ++).  
Самый простой вид итератора - это указатель. Указатель может ссылаться на элементы массива, также может передвигаться по ним с помощью операции ++.  
  
63. Можно ли реализовать последовательный доступ без итератора? В чем преимущества реализации последовательного доступа с помощью итератора?  
Да, можно, с помощью функций и хранения состояния внутри самого объекта.  
Удобство работы с разными объектами, т.к. у итератора как правило один интерфейс, в котором объявлены операции ++ и \*.  
  
64. Что играет роль итератора для массивов C++?  
Указатель.  
  
65. Что такое деструктор? Может ли деструктор иметь параметры?  
Метод класса, который вызывается для уничтожения объекта. Нет.  
  
66. Почему для классов контейнеров деструктор нужно писать явным образом?  
Потому что контейнеры, как правило, выделяют память для хранения объектов динамически. Соответственно, деструктор будет нетривиальным, т.к. потребуется пройтись по элементам коллекции и корректно очистить память для каждого элемента.  
  
67. Допускается ли перегрузка деструкторов?  
Да.  
  
68. Что такое "глубокое копирование" и когда в нем возникает необходимость?  
Глубокое копирование - копирование полей класса в том числе копирование объектов, на которые ссылаются поля-указатели (а не просто инициализация указателя тем же адресом). Необходимо выполнять при копировании "сложного" объекта (напр. объекта-контейнера), а также, как правило, объекта, который содержит инициализированные поля-указатели.  
  
69. Какое копирование осуществляет стандартный конструктор копирования?  
Неглубокое. Поэтому его нужно переопределять, например, для классов с полями-указателями или для классов контейнеров.  
  
70. Чем отличается копирование от присваивания?  
При копировании создается объект копия с глубоким копированием полей. Присваивание предполагает тривиальную инициализацию полей класса значениями из другого.  
  
71. Объясните, почему в операции присваивания требуется проверка присваивания самому себе?  
В некоторых случаях это может привести к "поломке" объекта, если мы присвоим объект самому себе. Пример из "More effective C++" Скотта Мейерса:  
class A  
{  
  A& operator=(const A& \_a);  
private:  
  char\* data;  
};  
A& A::operator=(const A& \_a)  
{  
  // Если поставить здесь if (this == &\_a), то всё будет ок.  
  delete [] data;  
  data = new char[strlen(\_a.data) + 1];  
  strcpy(data, \_a.data);  
  return \*this;  
}  
  
72. Можно ли в качестве операции индексирования использовать операцию вызова функции()? В чем её преимущества перед операцией []?  
Да, можно. Преимущество - количество аргументов больше чем один.  
A& operator[](int idx) { ... return \*this;}  
A& operator()(int idx, int idx2, ...) { ... return \*this;}  
  
73. Почему необходимо писать два определения операции индексирования? Чем они отличаются?  
Потому что должна быть операция индексирования для const объекта и для non-const объекта. Этим и отличаются.  
  
74. Дайте определение вложенного класса.  
Вложенный класс - класс, который определяется внутри другого класса. Соответственно - область видимости вложенного класса ограничивается внешним классом.  
  
75. Можно ли класс-итератор реализовать как внешний класс? А как вложенный? В чем отличия этих методов реализации?  
Да. Да.  
  
76. Может ли объемлющий класс иметь неограниченный доступ к элементам вложенного класса? А вложенный класс — к элементам объемлющего?  
Нет. Да  
  
77. Ограничена ли глубина вложенности классов?  
Нет.  
  
78. Можно ли определить вложенный класс внешним образом? Зачем это может понадобиться?  
Да. Если вложенный класс использует определение внешнего класса.  
  
79. Каким образом вложенный класс может использовать методы объемлющего класса? А объемлющий - методы вложенного.  
Вызывая их, например так:  
class A { int a; public: class B { public: B(A& a) { a.a = 2; } }; };  
Если хочется вызывать методы вложенного, то нужно внешний объявлять как friend внутреннего.  
  
80. Что такое "запредельный" элемент, какую роль он играет в контейнерах?  
Это элемент, использующийся для формирования условия остановки в цикле. Для условия остановки итератора.  
  
81. Объясните, по каким причинам трудно написать универсальный контейнер, элементы которого могут иметь произвольный тип?  
Причина - строгая типизация языка C++.