1. Основная идея ООП — использование вместо чёткого алгоритма одной цельной программы или разбиение её на насколько повторяющихся составляющих, взаимодействие некоторых объектов. Объекты — это экземпляры классов, абстрактных типов данных. Позволяют моделировать поведение реальных объектов и тот, кто решает задачу уже может действовать в терминах этой задачи, а не в терминах программирования.
2. Вся сущность ООП как раз в моделировании поведения объектов реального мира. Позволяет решающему задачу использовать термины этой задачи, а не программирования, что сильно облегчает задачу. Каждый класс имеет свои данные и методы для их обработки. Эти поля можно унаследовать, то есть можно создать класс, который будет точно таким же, как другой, но с некоторыми добавлениями. Это производный АТД или производный класс.
3. Концепции. Моделирование поведения реального мира, возможность создания своих типов данных, инкапсуляция (данные закрыты, методы открыты), повышение степени повторного использования кода благодаря наследованию и шаблонам, полиморфная обработка родственных объектов. Инкапсуляция — это принцип ООП, по которому каждый класс имеет закрытую часть с данными (свойствами) и открытую часть с методами их обработки (поведения). Открытую может использовать любая программа, закрытую только сам объект. Так защищаются данные внутри класса. Данные в классе менять нельзя, а вот методы изменять или дополнять всегда пожалуйста.
4. Концепции. Моделирование поведения реального мира, возможность создания своих типов данных, инкапсуляция (данные закрыты, методы открыты), повышение степени повторного использования кода благодаря наследованию и шаблонам, полиморфная обработка родственных объектов. Названия классов начинаются с большой буквы, названия методов с маленькой. В данных может храниться любая информация про этот объект. В том числе и его состояние, например положение в пространстве или температура, которая может часто меняться. Для этого используются методы, при помощи которых объект может менять свои свойства. Методы также могут вызывать другие методы других объектов, поэтому получается, что один объект действует на другой. Данные могут быть представлены в виде любых типов данных. Переменные, константы, структуры (в том числе динамические). Конструкторы — методы, которые создают объект и приводят его в состояние, в котором с ним можно работать. Деструкторы удаляют объект и очищают память, которую он использовал. Селекторы могут читать информация из объекта, обрабатывать её и выводить, но не менять. Модификаторы могут менять информацию внутри объекта.
5. Концепции. Моделирование поведения реального мира, возможность создания своих типов данных, инкапсуляция (данные закрыты, методы открыты), повышение степени повторного использования кода благодаря наследованию и шаблонам, полиморфная обработка родственных объектов. Классы используются для объединения схожих объектов и одинакового использования любого из них. Данные и методы записываются с маленькой буквы, названия классов с большой. Хорошо, если они отражают смысл, а не f1(), f2(), f3(). Наглядно представлены классы могут быть двумя способами. Как статическая модель, где отображаются все классы, используемые в программе и связи друг с другом и динамическая модель, где указывается момент времени, в который класс воздействует на другой класс. Например, первый класс может сначала поочерёдно воздействовать на два других и только после этого получить отдачу от первого воздействия. Объект — это экземпляр класса, который обладает всеми, заложенными в класс свойствами и способен на свойственное ему поведение. Изменение одного объекта никак не влияет на класс в целом. Объект сначала создаётся при помощи конструктора. Потому наполняется необходимыми данными через методы (в том числе может быть конструктор). Обработка производится также методами. На этой стадии объект активно используется, записываются данные в него, обрабатываются, удаляются и так далее. Разрушение вызывается деструктором. В нём освобождается память из-под всей информации которая была в нём.
6. Концепции. Моделирование поведения реального мира, возможность создания своих типов данных, инкапсуляция (данные закрыты, методы открыты), повышение степени повторного использования кода благодаря наследованию и шаблонам, полиморфная обработка родственных объектов. Конструктор — это особый метод, который используется для создания объекта и задания ему начальных значений. Он не возвращает никакого значения (даже не void) и называется также, как класс. Конструкторов как правило три. По умолчанию, без входных значений. Конструктор по значению принимает на вход информацию, которую сразу и записывает в поле данных. Копирующий конструктор. На вход принимает ссылку на объект того же класса. Он полностью копирует исходный объект. Деструктор действует как конструктор по умолчанию, но наоборот. Его цель уничтожить объект и освободить максимум памяти. Таким образом все динамические структуры массово удаляются.
7. Концепции. Моделирование поведения реального мира, возможность создания своих типов данных, инкапсуляция (данные закрыты, методы открыты), повышение степени повторного использования кода благодаря наследованию и шаблонам, полиморфная обработка родственных объектов. Перегрузка операторов это возможность использоваться обычные, стандартные (унарные или бинарные) операторы для обработки объектов класса вместо вызова функций и косплея на ассемблер. Делается это как раз при помощи слова operator. Используется в комплекте с каким-нибудь оператором вместо имени функции. Можно вызывать прямо по этому имени, но можно и использовать просто оператор. Ключевое слово friend используется как правило для отладки или для перегрузки операторов. Функция, обозначенная этим словом, имеет доступ к закрытым полям класса и может их изменять, таким образом почти являясь компонентной функцией для нескольких классов разом. Очень удобно, если нужно, к примеру, написать функцию для присваивания объекта одного класса объекту другого.
8. Концепции. Моделирование поведения реального мира, возможность создания своих типов данных, инкапсуляция (данные закрыты, методы открыты), повышение степени повторного использования кода благодаря наследованию и шаблонам, полиморфная обработка родственных объектов. Шаблоны — это трюк для сокращения длины кода и увеличения его повторного использования. Используется, когда нужно создать два класса, которые будут отличаться только типом данных, а всё остальное одинаковое. Например, если иногда нужна двойная точность в числе с плавающей точкой, но слишком редко, чтобы засорять память и использовать тип double просто всегда. Тогда можно сделать шаблон и использовать две копии класса, одна для double вторая для float. Ключевое слово template используется для объявления слова (как правило T) которым будет обозначаться тип данных, который будет меняться от разных типов класса. Контейнерные классы — это классы, объекты которых хранят другие объекты других (как правило) классов. Разумеется, не сами объекты, а указатели на них, это эффективнее.
9. Концепции. Моделирование поведения реального мира, возможность создания своих типов данных, инкапсуляция (данные закрыты, методы открыты), повышение степени повторного использования кода благодаря наследованию и шаблонам, полиморфная обработка родственных объектов. STL это стандартная библиотека шаблонов. В ней есть разные контейнеры, алгоритмы, итераторы и функциональные объекты. Стеки, очереди, двусторонние очереди и всё прочее. Функции также имеются самые различные, например поиск того или иного элемента или поиск повторяющегося элемента или удаление чего-нибудь и так далее.
10. Концепции. Моделирование поведения реального мира, возможность создания своих типов данных, инкапсуляция (данные закрыты, методы открыты), повышение степени повторного использования кода благодаря наследованию и шаблонам, полиморфная обработка родственных объектов. Наследование используется примерно для того же, для чего вообще почти всё из ООП. Для повторного использования одно и того же кода. Если какие-то свойства или поведение появляется в нескольких классах сразу, их делают родственными и все повторяющиеся свойства записывают в один базовый класс. И уже ему добавляют производные классы, которые наследуют от него все свойства и поведение, а уже поверх них накладывают свои собственные, уникальные именно для этого класса.
11. Концепции. Моделирование поведения реального мира, возможность создания своих типов данных, инкапсуляция (данные закрыты, методы открыты), повышение степени повторного использования кода благодаря наследованию и шаблонам, полиморфная обработка родственных объектов. Полиморфизм — это объявление нескольких методов одним и тем же именем, чтобы было удобнее использовать и проще запоминать. Простейший пример — перегрузка функции, какой-нибудь swap. Абстрактный базисный класс — это класс, объекты которого не создаются, это просто базис, который объединяет в себе характеристики, которые есть в целой куче других классов, но не имеют смысла сами по себе.
12. Концепции. Моделирование поведения реального мира, возможность создания своих типов данных, инкапсуляция (данные закрыты, методы открыты), повышение степени повторного использования кода благодаря наследованию и шаблонам, полиморфная обработка родственных объектов. Обработка исключительных ситуаций это один из методов устранения ошибок в работе программы, которые невозможно избежать другим образом. Например, если нужно запросить на ввод в консоли число у пользователя, проще всего поставить конструкцию tpy parse, чтобы программа не пыталась умножить на “У” или поделить на “Й”. Можно, конечно, запретить нажимать любые кнопки кроме цифр и заблокировать возможность использования вставки из буфера и так далее… Но вопрос уже в эффективности. Сама конструкция простая. Если внутри блока try происходит исключение, программа продолжает работать дальше после выполнения предусмотренных на этот случай действий.
13. Концепции. Моделирование поведения реального мира, возможность создания своих типов данных, инкапсуляция (данные закрыты, методы открыты), повышение степени повторного использования кода благодаря наследованию и шаблонам, полиморфная обработка родственных объектов. Преимущества: модульность приводит к простоте в работе программиста, уменьшает количество ошибок, ускоряет работу при одновременной разработке большой группой (каждый может заниматься своим классом независимо от других), возможность второго использования в родственных по смыслу проектах тех же самых классов без каких-либо вообще изменений. Недостатки: общее увеличение времени разработки, особенно если программист один, увеличение затрат времени на анализ и планирование, необходимость распределения ролей в команде, так как требуется почти (или полностью) независимая работа каждого человека, большая себестоимость. Язык ООП позволяет использовать ООП и не позволяет не использовать ООП. С++ был первым языком, в котором ООП появилось в привычном понимании, с объектами, классами, всеми концепциями и наследованием. В более ранних вариантах чего-то похожего на ООП, то не было объектов, то это были попытки в реактивное программирование, которое до сих пор нормально не реализовано, только как концепты и так далее. Конечно, объекты можно делать хоть на ассемблере, но именно поддержка на уровне языка появилась именно в С++. Плюс, С++ оказался ультра-популярным, поэтому все остальные языка после частично ровнялись именно на С++, поэтом объекты даже не будь они настолько удобными, всё равно бы расползлись во-всюда. С++ для профессиональной подготовки программистов полезен тем, что он практически не отличается от С, который называют самым низкоуровневым языком из высокоуровневых, но со всем функционалом современных языков. Его низкоуровневость позволяет обучающемуся программисту освоить все тонкости работы системы, всё выделение памяти, указатели и прочее. С ним не нужно учить десяток разных языков, каждый под свою ситуацию, С++ универсален и может делать почти всё на как минимум неплохом уровне и позволяет заглянуть внутрь процессов, некоторых из которых скрыты в части других языков. Плюс, он базируется на английском языке, который хотя бы немного знают практически все.
14. Классы — это пользовательские типы данных, которые кроме определённой информации (символ, число или указатель) имеют ещё и встроенные в них методы обработки этих данных. Компонентные данные — это данные, которые являются частью того или иного класса. По принципу инкапсуляции должны быть доступны только компонентным функциям. Компонентные функции — это такие функции, которые привязаны к тому или иному классу. Как правило используются для обработки данных в объекте, но ничто не мешает создать класс без данных и затолкать туда ворох функций, которые будут просто вызываться из класса, как будто это структурное программирование. Доступ бывает общий (public) или частный (private). Всё, что в общем доступе может быть использовано любой внешней функцией, в том числе main. То, что в частном доступе может быть использовано только компонентными функциями, в том числе общего доступа. При определении сначала идёт ключевое слово класс, потом имя класса и в фигурных скобках поля private со свойствами и public для методов их обработки. Объявление класса происходит в файле-заголовке, если он не шаблонный, то определение в файле исходного кода, то есть позже. Инициализация уже напрямую с объектом проводится. Объекты, объявленные как константы могут использовать только функции-селекторы.
15. Область видимости класса начинается и заканчивается там, где подключены в виде заголовочных файлов. Оператор двойного двоеточия позволяет пользоваться полиморфизмом, когда у нескольких классов могут быть одинаковые по лексике методы и отличить какой будет использоваться, или при определении компонентной функции. Например, унарная форма позволяет пользоваться глобальной переменной i даже если используется другая локальная переменная с тем же именем. Бинарная позволяет компилятору определить, к какому классу принадлежит та или иная функция, если их несколько в области видимости с одинаковым названием. Постоянно используется для определения функций вне заголовочного файла. Вложенные классы — это классы, которые объявлены внутри другого класса и могут быть использованы только им. То есть, используется в поле свойств. Часто используется в очень крупных классах, когда некоторые его части могут быть востребованы очень редко и эффективнее просто разбить класс на несколько составляющих.
16. Компонентными данным может быть фактические любые данные, любого типа, встроенного или пользовательского. Используются для описания класса, его свойств и отличительных черт, которые нужны для решения задачи. Объявляются внутри блока private, то есть не могут быть изменены ничем, кроме собственных компонентных функций. Статические данные — это такие данные, которые могут быть использованы любым объектом класса и для всех объектов они одинаковы, так как привязаны только к самому классу. Кстати, к ним имеется доступ даже до создания хотя бы одного объекта, прямо из класса. Константные статические данные ровно то же самое, только их нельзя менять. Удобно для экономии памяти, если нужна какая-то константа для вычислений, лучше вынести именно туда, если объектов создать много, то получится небольшая экономия памяти. Изменяемые компонентные данные можно изменять (о, неожиданность) при этом даже если объект объявлен как константа.
17. Компонентные функции используются для симуляции реального поведения объектов. В теории. На самом деле используются для изменения данных тем или иным способом, обработки и вывода этих данных, а также упрощения использования класса программистом. Функции бывают селекторами и модификаторами. Первые могут считывать и обрабатывать информацию из объекта и выводить результаты. Модификаторы могут её ещё и менять.
18. Как правило, практически все функции определяются вне тела класса, чтобы сэкономить дополнительные строки. Иногда функции могут быть определены прямо там, если функция короткая, это чуть быстрее писать. Однако, удобство в последующем использовании на стороне определяемых вне класса функций. Потому что при определении функции можно использовать простые названия для входных данных, которые удобны для именно этой маленькой задачи. А во время определения использовать длинные, но понятные, по которым легко понять, что именно запрашивается на вход такой функции. Синематика у встроенных простая настолько, насколько возможно. Просто определение функции в поле public как будто она глобальная. У внешнего определения нужно отдельно прописать область видимости класса, чтобы эта функция была видна только классу и/или его объектам.
19. Указатель this это особый указатель на объект, через который вызывается компонентная функция. Статические такого указателя не имеют, просто потому что они привязаны к классу, а не к объекту, а у константных этот указатель тоже константный, поэтому ничего кроме mutable нельзя изменить в объекте без дополнительных манипуляций. И static и const просто указываются в перед именем функции. Если очень хочется, то даже одновременно. Определяются и объявляются они как самые обычные компонентные функции, объявление в теле класса, определение можно вынести в отдельный файл с указанием расширения области видимости. Static функции используются для редактирования или ещё какого-то использования данных, тоже отмеченных как static ещё до создания объекта этого класса. А функции const это селекторы, защита от случайного изменения в объекте и, например, использования внутри селектора функции модификатора. Также, только константные функции можно использовать из константных объектов, потому что менять их нельзя. В целом, константы и статики пихать во всюда, куда можно.
20. Конструктор это особая компонентная функция, которая создаёт объект и вызывается тогда и только тогда, когда этот объект надо создать. В целом, часть конструктора находится в поле данных этого класса, когда объявляются, определяются и часто инициализируются его свойства. То, что нельзя сделать там по той или иной причине, выносится в конструкторы. Есть три вида конструкторов, конструктор по умолчанию, который просто создаёт объект, конструктор по значению, который создаёт объект с заданными свойствами и копирующий конструктор, который создаёт полную копию необходимого объекта. Первый и третий компилятор может предоставить сам, по умолчанию просто задаст данные объекту из поля данных определения класса, а копирующий побитово скопирует нужный объект, что приведёт к двойному указанию, если указатели есть. Если их вдруг нет, то сработает как надо. По значению хоть тресни, надо самому писать. Конструкторы по значению можно перегружать сколько угодно раз, давая пользователю возможность загрузить, к примеру только часть данных, остальные оставляя по умолчанию. Также есть ещё конструкторы преобразования, которые по своей сути похожи на копирующие, только переносят данных из объекта другого класса, например ничто не мешает создать такой конструктор, чтобы он создавал очередь копию двунаправленной очереди или вообще стека.
21. Явное описание конструктора это просто описание функции, которая имеет точно такое же имя, как и класс. Определение можно разместить прямо в теле класса, иногда позволяет выйграть немного производительности, но приводит к сложностям в компиляции с крупными проектами, потому что тогда эта реализация будет перекомпилироваться в каждый использующий этот класс проект отдельно, а если реализацию вынести в файл реализации достаточно перекомпилировать его одного и всё заработает сразу же. Конструкторы можно использовать только как конструкторы и нельзя вызывать нигде, кроме как в конструкторе. И даже в другом конструкторе вызывается конструктор не как обычная компонентная функция, его нужно вызвать до фигурных скобок определения конструктора. Иначе по завершению этого конструктора выйдет из области видимости и созданный объект, что приведёт к вызову деструктора. Однако, конструировать можно не только по значению или копию объекта этого же класса, но и на основе объекта другого класса, таким образом выполняя преобразование типа данных.
22. Конструкторы часто инициализируют объекты, например если в поле данных есть какие-то значения по умолчанию, их можно инициализировать прямо в теле цикла, а вот если какая-то структура, то надо ей ещё выделить память, что делается в конструкторе. После работы конструктора объект уже полностью готов к работе и ничего дополнительного не нужно делать, чтобы записывать в него информацию или как-то пользоваться уже имеющейся. Конструкторы также можно применять для конструирования объекта на основе другого объекта даже другого класса, что позволяет, фактически, выполнять преобразование типов, причём одновременно с инициализацией нового объекта. Удобно, сокращает жизнь на пару строк кода.
23. Конструкторы по умолчанию — это конструкторы, которые ничего не принимают на вход. Компилятор сам выдаст такой, если его не сделает программист и тогда такой конструктор просто выполнит весь код (кроме объявления функций) из тела самого класса. Объявит нужные переменные, проинициализирует те, которые были проинициализированы и будет готовый объект. Если в классе нет динамического распределения памяти, то такой способ имеет место быть и даже удобен. Если же его описывать, то нужно сначала объявить конструктор в теле цикла, потом (рекомендуется) этот самый конструктор определить в исполняемом файле. У конструкторов нет возвращаемого типа данных, даже void, он просто работает и всё. Также, всегда необходимо при определении показать, часть какого класса этот конструктор, даже несмотря на одинаковое имя функции и класса, компилятору этого мало.
24. Конструкторы копирования — это конструкторы, которые на вход принимают ссылку на объект того же класса, который необходимо создать. Также, конструктором копирования, в принципе, можно назвать конструктор, который приводит один тип данных к другому, потому что тоже принимает на вход ссылку да объект. Используется для, о неожиданность, копирования объектов. Задаются явно или используется по умолчанию, когда компилятор просто копирует побитово объект, устраивая парад двойных указаний, как оператор присваивания.
25. Конструктор вызывается всегда, когда нужно создать новый объект. В зависимости от переданных параметров выбирается вид конструктора. Если нет параметров, используется конструктор по умолчанию. Конструктор можно вызвать внутри другого конструктора, для этого нужно сразу перед фигурными скобками в определении нового конструктора через двоеточие вызвать необходимый конструктор. Например, часто используется в копирующем. Явный вызов конструктора — это вызов со скобочками, неявный это просто определение новой переменной (вернее объекта) типа класса.
26. Деструктор — это компонентная функция, обратная конструктору. Он используется для очистки выделенной динамически памяти. Также, после срабатывания деструктора объект выходит из области видимости программы и больше не может быть использован. Деструктор имеет точно такое же имя, как и класс, объект которого он удаляет, но с символом ~ в начале. На вход не принимает никаких значений. В конце любой функции, где создавались объекты, вызываются деструкторы для всех этих объектов, но можно и явно его задать, чтобы очистить память заранее.
27. Описание деструктора может быть как явным, так и по умолчанию, если явного не задано. Как правило, описывается деструктор в теле класса, а определяется уже в исполняемом файле. Разница ровно такая же, как и для любой другой компонентной функции. Иногда ускоряет работу, но создаёт дополнительные проблемы при компиляции многих проектов с использованием этого класса. У деструктора, как и у любой другой компонентной функции имеется всегда как минимум один параметр, указатель на используемый объект this. Но, в нём можно делать и другие параметры, если это необходимо. К примеру, если есть два и более способа удаления объекта.
28. Вызывать деструктор можно явно или он будет вызвать автоматически после выхода из области видимости функции, где был создан. В целом, можно спокойно создавать программы, явно не вызывая деструкторов. Однако, если объекты создаются в крупной функции и их там много это может привести к очень большому расходу памяти. Да, она освободится по завершению работы функции, но освобождать её можно и заранее.
29. Специальный полиморфизм — это полиморфизм, то есть называние, одним и тем же словом, разных вещей, который используется в специальных случаях. Его можно (и нужно) использовать для преобразования АТД, то есть пользовательского класса к встроенному типу. Конечно, это не всегда имеет смысл, но может пригодиться. Например, если АТД это класс столов, а для задачи нужно посчитать кроме всего прочего их общий вес, то конструкцию int + table \* 5 вполне нормально перегрузить, чтобы из класса table бралось определённое поле и использовалось как целое число.
30. Специальный полиморфизм — это полиморфизм (придача одной команде или оператору разных значений) который используется в специальных ситуациях. Перегружаются очень часто именно функции. Выбирается один их вариантов перегрузки функции по очень простому алгоритму. Поиск совпадений по списку входных параметров. И ничего больше тут не выбирается, это не преобразование данных, алгоритм прост как три копейки.
31. Дружественный функции — это функции, которые могут забивать на принцип инкапсуляции и доставать абсолютно любые данных из любого объекта, которым пользуется. Основное применение — перегрузка операторов. Например, если нужно перегрузить бинарный оператор от двух объектов разных классов. В такой ситуации можно обойти и без них и обойти открытой частью второго класса, а вот если первый оператор встроенный тип данных или какой-то внешний, к которому никак не хочется подступаться, то совсем уже другое дело. Например, оператор вычитания из целочисленной переменной объекта или перегрузка оператора вывода на консоль через std::cout. Но, так как эти дружественные функции нарушают принцип инкапсуляции, за удобством кроется и некоторая опасность. Частое применение таких функций может привести к дополнительным ошибкам в коде, которые будет ещё и очень сложно найти, ведь ломается какой-то объект, а функция которая его ломает находится чёрт знает где.
32. Объявление дружественной функции отличается от объявления обычной глобальной функции из структурного программирования только служебным словом friend, которое ставится перед объявлением. У неё нет никакого указателя на себя, поэтому для перегрузки бинарного оператора, к примеру, требуется передать два операнда. Возвращаемое значение. Если очень хочется, можно изгаляться и использовать дружественные функции, как компонентную функцию какого-нибудь класса, в которой не будет указателя на себя. Вдруг пригодится такой фокус. Но, если объявлять функцию как член какого-то класса, то это придётся сделать и во всех других классах, которые она собирается использовать.
33. Перегрузка операторов выполняется при помощи функций, называющихся operator. В зависимости от того, какой оператор нужно перегрузить, количество входных данных будет сильно разным. Например, для унарного оператора хватит всего одного. Все виды приоритетов и порядков выполнения полностью сохраняются, как это происходит и с операторами для встроенных классов, так если теоретически перегрузить все виды операторов для какого-то класса, то он ничем не будет отличаться от встроенных. Перегружать можно практически все операторы, за исключением точки, точки со звёздочной, тернарного оператора, указания области видимости и sizeof. Все остальные — всегда пожалуйста. Ну, а ещё нельзя перегружать операторы встроенных типов данных, мы не в C#.
34. Перегружать можно как унарные, так и бинарные операторы. Если это компонентные функции, то унарные операторы будут без явно заданных входных данных, а дружественные или просто внешние функции будут иметь один. Также бинарные операторы в компонентном виде будут иметь всего один, второй операнд. Ну а дружественным или внешним нужно передать оба.
35. Перегрузка этих операторов никак не отличается от перегрузки любых других операторов. Просто перегрузка и всё. Единственное, её очень удобно использовать в копирующем конструкторе. Достаточно вызвать таковой и просто присвоить ему значение копируемого объекта. Однако, оператор присваивания уже есть по умолчанию, и, если его не перегрузить, от всё равно будет работать. Правда присваивание может работать неверно, особенно если есть указатели, динамическое выделение памяти, массивы, да почти что угодно. Оператор индексирования не может быть дружественной функцией, только компонентной. При этом, если оператор будет возвращать новый объект этого класса, то его можно только извлекать. А вот если возвращает ссылку, то такая штука станет модификатором и функционал расширится.
36. Перегрузка операторов ввода-вывода. Как правило, для это используется дружественная функция, которая позволяет перегрузить операторы с необходимыми операндами std::cout << class obj. Так как первый операнд должен быть не объектом класса, для которого перегружается оператор, а другой, то используется только дружественная функция, использование нормальных компонентных невозможно.
37. Можно использовать для самой обыкновенной индексации, вот только позволяет использовать несколько параметров. То есть, двумерные или трёхмерные массивы не предел, хоть 10 мер можно, нужно только указать входные параметры, из всех операторов ни у какого другого нет таких фокусов. Хотя под вопросом, имеет ли смысл, ведь можно использовать и просто так обычную функцию, синтаксис вызова фактически не поменяется.
38. Просто так позволяет послать куда подальше принцип инкапсуляции, потому что хочется. Ни что теперь не мешает обратиться к члену объекта где угодно. По сути своей формально ничего не нарушается, ведь это перегруженная функция-оператор, работает как геттер и сеттер в одном флаконе.
39. Позволяет делать секс с конструкторами и деструкторами. В целом, как правило, перегрузка оператора new или delete позволяет автоматически добавлять пару (или много) строк кода, которые будут вызываться перед тем, как вызовется нормальный, обычный delete. Перегружать можно, само собой все их виды, хоть простые new и delete хоть с []. В остальном, оператор ведёт себя точно также как остальные операторы. В параметры передавать размер удаляемого объекта.
40. Шаблоны классов и функций задаются при помощи ключевых слов template <T>, где T теперь будет показывать тот тип данных, который может использоваться. По умолчанию в этот Т можно затолкать абсолютно всё, что угодно, но если объект будет пытаться вызвать функцию какого-то класса, а на вход подадут тип данных, в котором такой функции нет… ошибка компиляции. В этом причина полиморфной обработки, очень хорошо помогает, когда геттеры, сеттеры и прочее одинаково пишется и можно смело толкать всё, что угодно в какой-нибудь шаблонный класс-контейнер. Особенно хорошо в C#, где даже у встроенных типов имеется не мало методов. Но, основная цель шаблонных функций и классов в том, что можно использоваться один и тот же код для одинаковых функций и классов, которые отличаются только типом данных, которые сейчас обрабатывают.
41. Шаблоны классов используются для экономии кода, повышения его повторной использоваемости и уменьшения количества ошибок, связанных с большим количеством копирования кода с минимальными отличиями. Используется тогда, когда два класса работают абсолютно одинаково, но только с разными типами данных. Тогда их можно объединить в один. И, особенно здорово это работает если заранее неизвестно, какой тип данных будет использоваться. Таким образом шаблоны становятся просто незаменимыми. Объявляется это дело ключевыми словами template <T>, где Т обозначает тип данных, который будет разным. Далее во всём классе Т будет использоваться как такой универсальный тип данных. И, в целом, ничто не мешает вызвать метод какого-нибудь АТД а потом заместо Т подставить int. Если объект, где используется вместо Т int не будет использоваться ни одной компонентной функции от Т, то всё будет даже работать. Статические элементы будут напоминать программисту о том, что шаблонные классы — это не один класс под несколько типов данных, а много разных классов, код которых практически идентичен. Поэтому и статические элементы будут у каждого свои.
42. Шаблоны функций в использовании практически никак не отличаются от перегруженных функций, кроме того, что у перегруженных разный код под разные типы данных, а у шаблонных он один и тот же. Но, ничто не мешает перегрузить шаблонную функцию, например один параметр её будет шаблонным типом Т, а второй может быть или не быть. И будет две шаблонные функции с одинаковыми именем, ведь перегруженные функции это просто две функции с одинаковыми именами, а имена так и так будут благополучно забыты при компиляции.