Список вопросов для подготовки к экзамену по ООП:

**Что такое ООП? На чем делается основной акцент при разработке программ с использованием ООП?**

*ООП* - предоставляет возможность создавать объекты, которые объединяют характеристики и поведение в самостоятельный союз, которые можно многоразовое использовать.

*Наследование* – образование новых классов на основе ранее созданных со всеми его свойствами.

*Полиморфизм* – вызов методов из дочерних классов (один интерфейс и много реализаций).(Перегрузка функции)

*Инкапсуляция* – это возможность закрывать данные и методы класса от внешнего вмешательства, и потом можно вносить изменения.

*Абстрагирование*– это умение выделить только нужные характеристики и поведение.

**Что такое объект? *Объект* – это переменная (экземпляр) класса.**

* Из чего он состоит?

Состоит из *характеристики и поведения.*

* Что такое поля и методы объекта?

*Поля(характеристика) класса* – экземпляры(переменные).

*Методы(поведение) класса* – это его функции.

**Что такое класс? В чем отличие этого понятия от понятия объекта?**

*Класс* – это пользовательский тип данных, который описывает некоторые объекты предметной области характеристики и поведения.

**Что такое инкапсуляция? Зачем она нужна?**

*Инкапсуляци****я*** – это один из приципов ООП, которая предоставляет возможность закрывать данные и методы класса от внешнего вмешательства.

Она нужна для того, чтобы мы могли абстрагироваться от реализации и работать с интерфейсом. Так как, если мы меняем реализацию не сломался интерфейс.

Интерфейс предоставляет нам методы и поля, для работы с классом, а внутри содержит инкапсулированные поля и реализацию методов. Так же интерфейс может предоставить нам доступ к реализации для получения или изменения значений.

* Что такое интерфейс и реализация класса?

*Интерфейс* — это клиентский код - public.

*Реализация* - это все, что private, protected.

Что мы выигрываем как разработчики класса?

**Что такое спецификаторы доступа?**

***Спецификатор доступа*** определяет, кто имеет доступ к членам этого спецификатора.

* Какие спецификаторы доступа вам известны?

- *спецификатор public* делает члены открытыми.

- *спецификатор private* делает члены закрытыми.

- *спецификатор protected* открывает доступ к членам только для дружественных и дочерних классов.

* В чем отличие структур и классов?

1. Структура от классов отличается спецификатором доступа.

2. Если спецификатор наследования не указывать то для класса по умолчанию используется private, а для структур public.

* Разница между private и protected с точки зрения инкапсуляции?

**Что такое инспекторы и модификаторы?**

*Модификаторы* – это методы, которые позволяют устанавливать значение в поля классов и именуются с префиксом (set).

*Инспекторы* – это методы, которые позволяют получить значение из поля классов и именуются с префиксом (get).

**Что такое конструктор? Зачем нужны конструкторы?**

*Конструктор* – это особый тип метода класса, который автоматически вызывается при создании объекта этого же класса.

Конструкторы используют для инициализации переменных – членов класса значениями, которые предоставлены по умолчанию\пользователем, или для выполнения любых шагов настройки, необходимых для используемого класса.

* **Каков синтаксис конструкторов?**

- конструкторы всегда должны иметь то же имя, что и класс (учитывая верхний и нижний регистр).

- конструкторы не имеют типа возврата.

* **Когда вызывается конструктор?**

Предназначены для выполнения инициализации.

* **Что такое конструктор по умолчанию?**

Конструктор по умолчанию – это конструктор, который не имеет параметров (или содержит параметры, которые все имеют значения по умолчанию).

* **В чем удобство перегрузки конструкторов?** Конструктор может быть перегруженным, если у класса несколько конструкторов с разным списком параметров.
* **Что такое деструктор? Зачем нужны деструкторы?**

*Деструктор* – это специальный тип метода класса, который выполняется при удалении объекта класса. Деструкторы предназначены для очистки памяти после конструкторов.

-динамическая память

-файл

-соединение по сети.

* **Каков синтаксис деструкторов?**

- деструктор должен иметь то же имя, что и класс, со знаком тильда в самом начале.

- деструктор не может принимать аргументы.

- деструктор не имеет права возврата.

* **Когда вызывается деструктор?**
* Деструктор вызывается при уничтожении объекта в классе.
* В каких случаях не обойтись без деструктора?

Когда есть динамическая память, файл или соединение по сети.

* **Какой спецификатор доступа должен быть у конструкторов и деструкторов?**

Технически можно создавать конструктор с любым спецификатором доступа, но если объявить конструктор закрытым или защищенным, то создавать объект вне класса будет нельзя.

* **Каков порядок вызова конструкторов и деструкторов?**

*Порядок вызова конструктора*: при создании объекта производного класса в начале вызывается конструктор базового класса для инициализации наследованных полей, а затем вызывается конструктор производного класса для инициализации собственных полей.

*Порядок вызова деструткора:* в обратном порядке.

* **Что такое инициализаторы? Где они используются?**

*Инициализаторы* – это конструкции которые позволяют задать начальное значение поле класса до вызова конструктора.

Необходимо использовать в тех ситуациях, когда поле класса должно быть проинициализирована при создании. Если полем класса является константа, ссылка или объект класса у которого нет конструктора по умолчанию. А так же инициализаторы используются при наследовании если у базового класса нет конструктора по умолчанию.

* Что такое указатель this? Зачем он нужен?

This – это скрытый константный указатель, содержащий адрес объекта, который вызывает метод класса.

* Какую проблему решает this?

Если у нас есть конструктор(или метод), который имеет параметр с тем же именем, что и переменная-член, то устранить неоднозначность можно с помощью указателя this.

* Как и где можно пользоваться указателем this?

Создается при вызове нестатических методов класса и получает адрес вызывающего объекта.

* Семантика копирования
* Зачем нужен конструктор копирования? В каких ситуациях без него нельзя обойтись?

Конструктор копирования нужен в тех случаях, когда мы работаем с динамической памятью. По – умолчанию конструктор копирования выполняет побитовое копирование Но если имеет дело с указателем или ссылкой, то при копировании по умолчанию скопируется только указатель, а не ресурс, и у нас получиться битый указатель.

Конструктор вызывается в 3 ситуациях, а копирование происходит в 4 ситуациях

* Когда создается объект на основе уже существующего типа.
* Когда передаем параметры в функцию по значению
* Когда функция возвращает значение
* Когда присваиваем один объект в другой (относиться к присваиванию)

Копирование реализуется двумя способами:

Оператор присваивания и конструктор копирования

Присваиваем один объект класса другому объекту уже существующему объекту.

Из правого операнда копируется значение в левый операнд.

Нужно проверить на само присваивание, очистить левый операнд, и создается ссылка на левый операнд, для того чтобы можно было использовать в выражении.

Семантика переноса появилась в с++ 11 версии. И компилятор сам различает где нужно вызывать копирование, а где перенос, оснуется на том, какой тип ссылки он получает. L-value \ R-value.

Для переноса и копирования реализуется две версии

Для переноса r-value (справа от занка равно, это временный объект, потому что неявно вызывается при возвращет функцию, и нельзя получить адрес на это.)

Для копирования l-value константные ссылки чтоб использовать слева от знака равно.

* Что такое перегрузка операций? Для чего она нужна?

Перегрузка операции - это возможность определить стандартные операции для своего класса. Идея заключается в том, чтобы интегрировать пользовательские типы данныхв язык.

Перегрузка операции нужна для шаблоных функций и методов класса. Если не было бы перегрузки операторов, то в шаблоны нельзя было подставлять пользовательский тип данных.

Перегрузка операции определяется с помощью функции или метода класса с именем “ operator знак”.

* Каков синтаксис перегрузки операций?

Тип оperator знак(параметры)

{блок кода}

* С помощью какой конструкции языка перегружаются операции?

Перегрузить только существующие операции. Изначалько количество операндов, поддерживаеміх операторов, изменить невозможно.

* Все ли операции можно перегрузить как методы и функции?

Большинство операций можно перегрузить 2 способами.

Глобальной функцией и методом класса.

Перегрузка методом класса:

1.Оператор присваивания =

2.Оператор вызова функции()

3.Оператор индексации[]

4.Выбор члена класса через указатель->2

5. Оператор delete

6. Оператор new

7. Оператор приведения типа

* Есть ли операции, которые можно перегрузить только глобальными функциями?

Если левый операнд бинарной операции является объектом не нашего типа данных то перегрузить операцию методом невозможно. Так как метод всегда вызывается для левого операнда.

* Какие операции нельзя перегружать?

Есть операции которые нельзя перегрузить:

1.Тернарный оператор

2. Оператор sizeof

3. Оператор разрешения области видимости(::)

4. Оператор выбора члена(.)

5. Указатель, как оператор вызова члена(\*)

* Можно ли перегрузить операции для стандартных типов, например, для int?

Нельзя перегрузить для стандартных типов данных хотя бы один из операндов должен быть пользовательского типа.

* Можно ли ввести свои собственные новые операции?
* Существует ли в C++ неявная перегрузка операций?
* Каков синтаксис перегрузки операции инкремента и декремента?

Инкремент и декремент имеют 2 формы: префиксную и постфиксную.

Префиксный вариант должен изменять свой вариант и возвращать модифицированное значение.

Постфиксный вариант должен изменять свой операнд, но возвращать первоначальное значение.

* Чем отличается перегрузка постфиксного и префиксного вариантов перегрузки операции инкремента и декремента?

Для того, чтобы при перегрузке операции различались формы префиксной и постфиксная передается фиктивный параметр int.

* Что такое друзья класса?
* Что такое дружественные функции? Зачем они нужны? Каков синтаксис объявления дружественных функций?

*Дружественная функция* – это функция которая имеет доступ к закрытым членам класса, как если бы она сама была членом класса. Это позволяет функции(классу) работать в тесном контакте, с другим классом, не заставляя другой класс делать открытыми свои закрытые члены.

*Проблема, которую решает дружественная функция:*

Использование дружественных классов и дружественных функций, с помощью которых можно будет предоставить доступ к закрытым данным класса.

*Синтаксис:*

Для объявления дружественной функции используется ключевое слово friend, перед прототипом функции, которую вы хотите сделать дружественную.

* Что такое дружественные классы?

Один класс может быть дружественным другому классу. Это откроет всем членам первого класса доступ к закрытым членам второго класса.

* Что такое дружественная перегрузка операций?

Поскольку наша перегруженная функция является дружественной классу, то мы можем напрямую обращаться к закрытому члену.

* Что такое статические поля класса?

Статическое поле позволяет реализовать хранение данных, которые будут существовать вплоть до выхода из функции меин.

* Каков синтаксис объявления статических полей класса?

Ключевое слово static, и тип данных

* Как происходит обращение к статическим полям класса?

Статические поля нужно объявлять в глобальную область видимости.

Обращаться к статическим полям можно через имя класса или через объект.

* Что такое статические методы класса?

Статические методы класса – метод, не имеющий доступа к состоянию объекта, то есть к переменной this.

Слово статистический используют в том смысле, что статистические методы не относятся к динамике объекта, не используют и не меняют его состояния.

Они нужны для работы со статическими переменными – класса. Для работы с ними не требуется создавать объекты класса.

* Каков синтаксис объявления статических методов класса?

Ключевое слово static, и тип данных

* Как происходит вызов статических методов класса?

Статические методы класса могут вызываться через статические объекты и имя класса, они не получают адрес объекта, для которого вызваны, следовательно, внутри такого метода нет указателя this и нельзя обращаться к экземплярным полям.

Экземплярные поля могут вызываться только для объекта класса (Они получают адрес этого объекта в указатель this и могут работать с экземплярными полями объекта.)

* Что такое динамическая структура данных?

Динамическая структура данных – это любая структура данных, занимаемый объем памяти которой не является фиксированным.

* Что такое стек? Каковы его принципы работы?

Стек – это структура данных типа LIFO (Последним пришел, первым ушел).

Последний элемент, который находится на вершине, первым уйдет из него.

* Что такое очередь? Каковы её принципы работы?

Очередь – это структура данных, которая построена по принципу LILO (Последним пришел, последним вышел).

Реализация очереди:

Через одномернный масив, односвязный список и ооп класса.

* Что такое односвязный список? Какова его структура? Что такое голова списка?

Односвязный список – это динамическая структура данных, состоящая из узлов. Каждый узел будет иметь какое-то значение.

Основное назначение связного списка – предоставление механизма для хранения и доступа к произвольному количеству данных.

Как работает односвязный список – каждый узел однонаправленного списка содержит одно поле указателя на следующий узел. Поле указателя последнего узла содержит нулевое значение.

Голова списка – это начало списка, через который мы обращаемся.

* Что такое двусвязный список? Какова его структура? Что такое хвост списка?

Двусвязный список содержащий два указателя на следующий элемент и на предыдущий.

Поле указателя на следующий узел последнего узла содержит нулевое значение. Поле указателя на предыдущий указатель первого узла (корня списка) так же содержит нулевое значение.

* Что такое шаблоны классов?

Шаблон – это заготовка на основе которой компилятор создает функцию или класс. Используется для описания алгоритмов, который отличаются лишь типом данных, создаваемых переменных.

* Каков синтаксис объявления шаблонов классов?

// Синтаксис

template<typename T>

void MySwap(T&a,T&b)

{

T tmp = a;

a=b;

b = tmp;

}

* Как по шаблону класса создать класс?
* Почему шаблон относится к полиморфизму?

Шаблон реализуют принцип статического полиморфизма.

Полиморфизм позволяет при вызове операции с одним и тем же названием выбирать различные реализации, в зависимости от контекста вызова. Так как выбор реализации шаблона осуществляется на этапе компиляции, то полиморфизм здесь статический.

* Что такое агрегирование?

Агрегирование – это отношение между классами по принципу часть – целое. Позволяет нам создавать новый класс на основе уже существующего путем их включения.

* Агрегация

Агрегация – это отношение между двумя объектами, когда один объект имеет ссылку или указатель на другой объект. Оба объекта могут существовать и независимо, если объект целое будет уничтожен, то его объект часть нет.

* Композиция

Композиция – более строгий вариант агрегирования, когда включаемый объект может существовать только как часть контейнера. Если контейнер будет уничтожен, то и включенный объект тоже будет уничтожен.

* Что такое наследования? Для чего нужно?

Наследование - это принцип ООП, который позволяет создавать классы на основе уже существующих. Таким образом класс получает все поля и методы класса от которого он наследуется, а так же можно расширять поведение добавляя новые методы и поля.

* Каков синтаксис наследования классов?

*class Base{*

*...*

*};*

*class Derived : спецификатор доступа Base{*

*...*

*};*

* Что такое спецификаторы наследования? Как они влияют на наследование?

Спецификатор наследования указывает с какими спецификаторами доступа наследуются поля и методы базового класса.

* Наследуются ли конструкторы и деструкторы?

Конструкторы и деструкторы не наследуются.

* В каком порядке вызываются конструкторы при создании объекта унаследованного класса и деструкторы при его уничтожении?

Порядок вызова конструктора: при создании объекта производного класса в начале вызывается конструктор базового класса для инициализации наследованных полей, а после вызывается конструктор производного класса для инициализации собственных полей.

* От чего позволяет абстрагироваться наследование?
* Абстрагируемся от типа данных.
* Как происходит абстрагирование?

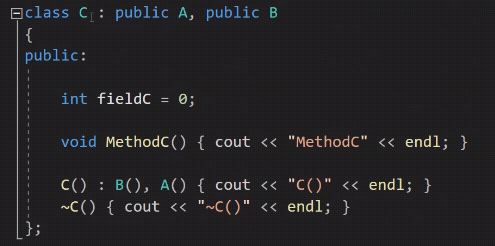
Создает указатель на базовый класс и присваивает ему объект Аклассу наследника.

Для чего так делать?

* Что такое множественное наследование?

Ромбовидное наследование – то есть класс унаследован более чем от 1 родителя.

* Каков синтаксис множественного наследования?



* Каков порядок вызовов конструкторов и деструкторов при множественном наследовании?

При создании объекта такого класса, вызываются все конструкторы его родителей, а затем и его собственные.

Порядок вызова конструктора определен порядком класса в списке наследования.

Деструкторы вызываются в обратном порядке.

* Как решается проблема конфликта имен при множественном наследовании?

В 2 базовых классах могут быть поля и методы с одинаковыми идентификаторами. Для доступа к полям и методам в производном классе нужно использовать операцию разрешения области видимости.

* Что такое виртуальный базовый класс? Каков синтаксис объявления виртуального базового класса?
* Что такое полиморфизм? В чем проблема?

Полиморфизм – один интерфейс и много реализации. Через указатель на базовый класс можно вызывать методы, которые объявлены в базовом классе.

* Виды полиморфизма?
* Динамический полиморфизм?
* Что такое раннее связывание?

По умолчанию связывание происходит на этапе компиляции и называется ранним связыванием. Из-за раннего связывания здесь вызовется реализации метода из базового класса, несмотря на то, что есть переопределенная версия в производном.

Дело в том, что компилятор определяет какую реализацию метода использовать по типу данных указателя, для которого он вызван. Дело в том, что объект на который ссылается указатель, может быть выбран на этапе выполнения.

* Что такое позднее связывание? Для чего оно нужно?

Для решения такой проблемы существует механизм позднего связывания . Такое связывания реализуется на этапе выполнения. Механизм позднего включения с помощью ключевого слова virtual.

Благодаря позднему связыванию реализуется полиморфизм.

* Что такое виртуальные методы? Каков синтаксис объявления виртуальных методов?

Производные классы обязана переопределить чистые виртуальные функции задав им реализацию таким образом, базовый класс объявляет интерфейс который обязаны реализовать все производные классы, таким образом работая через указатель на базовый класс есть гарантия, что методы будут реализованы.

Класс который имеет хотя бы одну абстрактную функцию называют абстрактными и объект такого класса создавать нельзя.

Если производный класс переопределит все чистые виртуальный функции то он станет абстрактным.

* Могут ли конструкторы быть виртуальными?

Конструкторы виртуальными быть не могут, так как при создании объекта класса мы всегда указываем конкретный тип данных, а, следовательно, указываем какой конструктор вызываем.

* Могут ли деструкторы быть виртуальными? Для чего нужны виртуальные деструкторы?

Деструкторы могут быть виртуальными так, как при удалении объекта через указатель на базовый класс по умолчанию будет вызываться деструктор базового класса, потому что для них тоже используется ранее связывание по умолчанию, и чтобы при удалении через абстракцию вызывался нужный деструктор они должны использовать механизм позднего связывания.

Так же деструкторы могут быть чисто виртуальными, но им все равно нужно предоставлять тела потому что при удалении объекта производного класса всегда вызывается деструктор базового класса.

* Что такое чисто виртуальная функция? Что такое абстрактный класс?

Наследование позволяет нам абстрагироваться от конкретного типа данных, создавая указатель, на базовый класс которому присваивается адрес объекта производного типа через указатель на базовый класс.

Можно вызывать только те методы, которые объявлены в базовом классе производные классы могут предоставлять поведение методов базового класса. но бывают ситуации когда в базовом классе должен быть объявлен метод.

Такие методы называются чистыми виртуальными функциями и абстрактными классами.

* Что такое исключение?

Исключения – это ошибка, которая происходит на этапе компиляции.

Необходимость обработки исключительных ситуаций?

При разработке класса или функций, мы не всегда знаем как обработать ошибку, но необходимо уведомлять о том, что ошибка произошла, для того, чтобы в клиентском коде можно было обработать

* Что делает оператор throw?

Этот оператор используется для сигнализации об ошибке.

* Что делает оператор try?

Обработка исключений с помощью блока. Тру код в котором возможна генерация исключений.

* Что делает оператор catch?

Этот оператор код обработки.

* Что такое необработанное исключение?
* Что такое раскрутка стека вызовов?
* Зачем нужна повторная генерация исключений?
* Что такое пространство имён?
* Для чего оно используется?
* Что делает оператор using?
* Могут ли быть вложенные пространства имен?
* Преобразование типов в С++
* Для чего используется оператор const\_cast?

При преобразовании проводит проверку на этапе компиляции и позволяет делать только допустимые преобразования.

* Преобразование численные преобразований друг друга
* Указателей друг на друга

Поддерживает те преобразования, который компилятор умеет делать неявно.

* Для чего используется оператор dynamic\_cast?

Этот оператор используется для преобразования полиморфных типов данных. То есть для указателя на классы, которые содержат хотя бы один виртуальный метод.

Выполняет проверку возможности приведения на этапе выполнения.

Если используется для приведения типов ссылок, то в случае ошибки будет генерировать bad\_cast. А если использовать для приведения указателей, то в случае ошибки возвращает 0.

* Для чего используется оператор reinterpret\_cast?

Преобразование без проверки. Может использоваться для преобразована указатель на указатель, указателю к целому и наоборот. Но применять только в полной уверенности.

* Для чего используется оператор static\_cast?
* Что такое STL?
* Основные понятия STL?
* Что такое контейнер?
* Для чего нужен итератор?
* Что такое алгоритм?
* Что такое функтор?
* Что такое предикат?