1.Основные идеи ООП: использование объекта в качестве основной компоненты программы и децентрализация управления, реализуемая представлением программы совокупностью взаимодействующих объектов

Особенностью объектно ориентированного программирования является то, что в отличие от процедурных языков, где ключевыми являются различные функции, а данным отводится так или иначе второстепенное место, при объектно-ориентированном программирование основную позицию в структурировании программ занимает объект, то есть сущность которая в себе инкапсулирует данные + необходимые функции для работы с этими данными. Структуру объектно-ориентированной программы можно было бы представить в виде дерева: то есть один объект содержит в себе другие объекты, другие объекты также содержат в себе какие-то объекты. Объектно ориентированное программирование обладает тремя ключевыми возможностями: 1. Инкапсуляция — это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя. Это свойство позволяет предоставить программисту только пользовательский интерфейс работы какого-либо объекта, дабы уменьшить возможность возникновения ошибки. 2. Наследование — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником или производным классом. В жизни это позволяет избавляться от дублирования кода, определяя в базовом классе большинство используемых данных и методов, а с помощью наследования расширять их. 3. Полиморфизм — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Это свойство позволяет использовать объекты, различные по своей природе, но обладающие общими интерфейсами, это свойство придает дополнительную гибкость при объектно-ориентированном программировании. Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций»

2. Сущность ООП: моделирование поведения объектов реального мира, наглядное представление объектов, создание и использование АТД, производные АТД.

Все эти модели являются абстракциями, представленными вычислениями в компьютере. Для описания таких моделей применяют понятие абстрактный тип данных, АТД (или сложный тип данных). Объекты являются переменными класса. Объектно-ориентированное программирование позволяет легко создавать и использовать АТД. Объектно-ориентированное программирование использует механизм наследования. Наследование выгодно тем, что позволяет получать производные типы из уже определенных пользователем типов данных. Классы Классы имеют ряд отличий от структур: •в объявлении используется ключевое слово class; •по умолчанию все члены класса имеют спецификатор доступа private, если не указано иное. Члены-данные структуры по умолчанию имеют тип доступа public, если не указано иное; •объекты классов всегда имеют таблицу виртуальных функций, даже если в классе не объявлено ни одной виртуальной функции. Структуры не могут иметь виртуальных функций; •к объектам класса можно применять оператор new, к структурам этот оператор применять нельзя; •классы могут наследоваться только от классов, структуры могут наследоваться только от структур. Производный класс наследует описание базового класса, делая ненужными повторную разработку и тестирование кода.

3. Концепции ООП. Инкапсуляция и расширяемость типов.

Инкапсуляция – это способность скрывать внутренние детали при предоставлении открытого интерфейса к определяемому пользователем типу. в C++, для обеспечения инкапсуляции используются определения класса и структуры (class и struct) в сочетании с ключевыми словами доступа private (закрытый), protected(защищенный) и public (открытый). Ключевое слово public показывает, что доступ к членам, которые стоят за ним, является открытым безо всяких ограничений. Без этого ключевого слова члены класса по умолчанию закрыты. Закрытые члены доступны только функциям-членам только своего класса. Защищенные члены класса доступны функциям-членам не только своего класса, но и классов-наследников. Открытые члены доступны любой функции внутри области видимости объявления класса. Закрытость позволяет спрятать часть реализации класса, предотвращая тем самым непредвиденные изменения структуры данных. Ограничение доступа или сокрытие данных является особенностью объектно-ориентированного программирования.

4. Концепции ООП. Семантика объекта: данные как отражение состояния объекта и методы как средства обеспечения взаимодействия объектов. Представление данных. Классификация методов: конструкторы, деструкторы, селекторы, модификаторы.

Методы по своим целям и предназначениям делятся: § Конструкторы - это методы, нужные для непосредственной инициализации начального состояния объектов. Данный метод вызывается при создании объекта, может также принимать различные параметры на вход, тем самым, конструируя различным образом объекты. § Деструкторы - это методы, которые вызываются непосредственно при уничтожении объектов. Предназначенны для освобождения различных занятых ресурсов удаляемым объектом, например, очистка памяти от внутренних переменных, разъединение задействованных подключений. § Селекторы- это специальные методы (по внешнему виду не отличающиеся от обычных методов) целью которых является предоставление доступа к данным внутри объекта, нужны как правило для того, чтобы предоставлять информацию в какой-то обработанной форме, либо, для того чтобы составить пару методу-сеттеру § Модификаторы - это методы которые изменяют состояния объектов, нужны для того, чтобы исключить непосредтвенный доступ пользователя объекта к данным внутри класса. Использование сеттера предоставляет возможность проверки устанавливаемых данных при установке значений, а также выполнение различных сопутствующих обработок необходимых для установки данных.

5. Концепции ООП. Классы объектов: назначение и семантика, классы и абстрактные типы данных, наглядное представление классов. Объекты как экземпляры классов. Основные действия с объектами: создание, инициализация, обработка, разрушение.

Классы предназначены для того, чтобы описывать переменные и методы, которые будут использовать объекты класса. Как правило класс представляет собой абстракцию некоторого реального объекта, в котором выделяются ключевые свойства этого объекта и исключаются не значимые в текущем контексте. С этой точки зрения объекты являются значениями данного абстрактного типа, а определение класса задаёт внутреннюю структуру значений и набор операций, которые над этими значениями могут быть выполнены. Наглядное представление класса можно, получить при построении диаграмм классов, где указываются списки методов и переменных классов, а также иерархии наследования. Объекты представляют собой сущности, описанные с помощью классов, где каждый объект имеет свой индивидуальный набор данных, список методов и адрес в памяти. В ООП с применением классов каждый объект является «экземпляром» класса, и других объектов не предусмотрено. То есть «экземпляр класса» в данном случае означает не «пример некоторого класса» или «отдельно взятый класс», а «объект, типом которого является какой-то класс».

Модификатор Операция, которая изменяет состояние объекта

• Селектор Операция, считывающая состояние объекта, но не меняющая состояния

• Итератор Операция, позволяющая организовать доступ ко всем частям объекта в строго определенной последовательности

• Конструктор Операция создания объекта и/или его инициализации

• Деструктор Операция, освобождающая состояние объекта и/или разрушающая сам объект

6. Концепции ООП. Конструкторы и деструкторы, основное назначение.

§ Конструкторы - это методы, нужные для непосредственной инициализации начального состояния объектов. Данный метод вызывается при создании объекта, может также принимать различные параметры на вход, тем самым, конструируя различным образом объекты. Функция-конструктор, являющаяся членом класса и имеющая имя, совпадающее с именем класса. конструктор не имеет типа возвращаемого значения. § Деструкторы - это методы, которые вызываются непосредственно при уничтожении объектов. Предназначены для освобождения различных занятых ресурсов удаляемым объектом, например, очистка памяти от внутренних переменных, разъединение задействованных подключений.

7. Концепции ООП. Перегрузка операторов и функций. Ключевые слова: friend, operator

Механизм друзей позволяет классу разрешать доступ к своим неоткрытым членам. Объявление друга начинается с ключевого слова friend и может встречаться только внутри определения класса. Механизм перегрузки операций позволяет обеспечить более традиционную и удобную запись действий над объектами. Для перегрузки встроенных операторов используется ключевое слова operator. Синтаксически перегрузка операций осуществляется следующим образом: тип operator @ (список\_параметров-операндов) { // ... тело функции ... } где @ — знак перегружаемой операции (-, +, \* и т. д.), тип — тип возвращаемого значения. Тип возвращаемого значения должен быть отличным от void, если необходимо использовать перегруженную операцию внутри другого выражения. Имеется два способа описания функции, соответствующей переопределяемой операции: § если функция задается как обычная функция-элемент класса, то первым операндом операции является объект класса, указатель на который передается неявным параметром this; § если первый операнд переопределяемой операции не является объектом некоторого класса, либо требуется передавать в качестве операнда не указатель, а сам объект (значение), то соответствующая функция должна быть определена как дружественная классу с полным списком аргументов. Перегрузка операций индексирования Переопределение операции () позволяет использовать синтаксис вызова функции применительно к объекту класса (имя объекта с круглыми скобками). Количество операндов в скобках может быть любым. Перегрузка унарной операции Если унарная операция перегружается как функция-член, то она не должна иметь аргументов, так как в этом случае ей передается неявный аргумент-указатель this на текущий объект. Перегрузка операций выделения памяти Операции создания и уничтожения объектов в динамической памяти могут быть переопределены следующим образом void \*operator new(size\_t size); void operator delete (void \*); Перегрузка операции присваивания Любой конструктор вызывается явно либо неявно в том случае, если необходимо создать новый объект какого-либо класса. Если необходимо осуществить присваивание, но поведение операции присваивания по умолчанию не устраивает, то операция присваивания может быть перегружена.

8. Концепции ООП. Шаблоны и обобщенное программирование. Ключевое слово: template. Контейнерные классы.

Шаблоны — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам. Обобщённое программирование –описании данных и алгоритмов, которое можно применять к различным типам данных. Шаблоны позволяют создавать параметризованные классы и функции. Параметром может быть любой тип или значение одного из допустимых типов. В отсутствие шаблонов программисту приходится писать отдельные функции для каждого используемого типа данных. Шаблон функции начинается с ключевого слова template, за которым в угловых скобках следует список параметров. Затем следует объявление функции. допускает использование class вместо typename: template< class T > Если в шаблоне класса или функции необходимо использовать один и тот же шаблон, но с разными параметрами, то используются параметры-шаблоны. Для вызова шаблонной функции, необходимо указать значения для всех параметров шаблона, после имени шаблона указывается список значений в угловых скобках. Контейнерные классы -- это универсальные шаблонные классы, предназначенные для хранения элементов заданного типа в смежных областях памяти.

9. Концепции ООП. Основные составляющие STL: контейнеры, итераторы, алгоритмы.

Контейнеры - это объекты, содержащие другие объекты. Существует несколько различных типов контейнеров. Например, класс vector, определяет динамический массив, класс queue создает двустороннюю очередь, а класс list обеспечивает работу с линейным списком. Эти контейнеры называются последовательными контейнерами, поскольку в терминологии STL последовательность представляет собой линейный список. Алгоритмыдействуют в контейнерах. Они включают возможности инициализации, сортировки, поиска и преобразования содержимого контейнеров. Многие алгоритмы работают с группой(или диапазоном) элементов внутри контейнера. Итераторы - это объекты, которые в той или иной степени являются указателями. Они позволяют циклически опрашивать содержимое контейнера практически так же, как это делает указатель для циклического опроса элементов массива. Существует пять типов итераторов.

Итераторы Разрешаемый доступ

Произвольного доступа (random access) Сохраняют и считывают значения; позволяют организовать произвольный доступ к элементам

Двунаправленные (bidirectional) Сохраняют и считывают значения; обеспечивают инкрементно-декрементное перемещение

Однонаправленные (forward) Сохраняют и считывают значения; обеспечивают только инкрементное перемещение

Входные (input) Считывают, но не записывают значений; обеспечивают только инкрементное перемещение

Выходные (output) Записывают, но не считывают значения; обеспечивают только инкрементное перемещение

10. Концепции ООП. Наследование: базовый и производный класс, иерархия классов.

Наследование — механизм языка, позволяющий описать новый класс на основе уже существующего (родительского, базового) класса. Класс, от которого произошло наследование, называется базовым или родительским . Классы, которые произошли от базового, называются потомками, наследниками или производными классами. иерархия классов в информатике означает классификацию объектных типов, рассматривая объекты как реализацию классов и связывая различные классы отношениями наподобие «наследует», «расширяет», «определение интерфейса».

11. Концепции ООП. Полиморфизм. Абстрактный базовый класс.

Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеетполя, методы и не может использоваться для непосредственного создания объекта. В полиморфизм подтиповпредполагает использование единого имени (идентификатора) при обращении к объектам нескольких разных классов, при условии, что все они являются подклассами одного общего надкласса (суперкласса). Полиморфизм подтипов состоит в том, что несколько типов формируют подмножество другого типа (их базового класса) и потому могут использоваться через общий интерфейс.

12. Концепции ООП. Обработка исключительных ситуаций, проверка утверждений.

Обработка исключительных ситуаций — механизм языков программирования, предназначенный для описания реакции программы на ошибки выполнения и другие возможные проблемы (исключения), которые могут возникнуть при выполнении программы и приводят к невозможности дальнейшей отработки программой её алгоритма. Область действия обработчиков начинается специальным ключевым словом try и заканчивается перед описанием обработчиков (catch). Обработчиков может быть несколько, один за одним, и каждый может указывать тип исключения, который он обрабатывает.

13. Концепции ООП. Преимущества и недостатки ООП. Основные свойства языка ООП. Место языка Си++ во внедрении и развитии средств ООП. Значение языка Си++ для профессиональной подготовки программистов.

Достоинства ООП: - «более естественная» декомпозиция программного обеспечения, которая существенно облегчает его разработку. - объектный подход предлагает новые способы организации программ, основанные на механизмах наследования, полиморфизма, композиции, наполнения. - механизмы позволяют конструировать сложные объекты из сравнительно простых. В результате существенно увеличивается показатель повторного использования кодов и появляется возможность создания библиотек классов для различных применений. Недостатки ООП обуславливаются следующим: - разработка библиотек классов и их использование требуют существенных трудозатрат. - не всегда легко определить, какие поля и методы фактически относятся к данному классу. - Код для обработки сообщения иногда «размазан» по многим методам (иначе говоря, обработка сообщения требует не одного, а многих методов, которые могут быть описаны в разных классах). Основной недостаток ООП - некоторое снижение быстродействия за счет более сложной организации программной системы. К основным свойствам объектно-ориентированных языков можно отнести то, что основными понятиями в ООП являются класс и объект. Класс является описанием на языке программирования какой-либо сущности Объект - это какая-либо сущность в адресном пространстве, которая строится на основании класса. С++ можно назвать языком программирования, который в наибольшей мере соответствует парадигме ООП. В него входят все основные концепции. Для профессиональной подготовки программистов язык C++ является важным по причине того, что он включает в себя все концепции + обладает ручным управлением памятью, что позволяет сформировать у программиста методику работы с ресурсами компьютера.

14. Классы: компонентные данные, компонентные функции, способы задания доступа, синтаксическая структура определения класса, объявление, определение и инициализация объектов, объекты объявленные как константы.

Компонентная функция должна быть обязательно описана в теле класса. В отличие от обычных (глобальных) функций компонентная функция имеет доступ ко всем компонентам класса. Функция–компонент класса имеет ту же область видимости, что и класс, к которому она относится. <имя класса> : : <имя компонентной функции> Класс в языке C++ создаётся следующим образом: class MyClass: public ParentClass // ParentClass — класс-предок, если таковой имеется { public: // элементы в этой секции доступны из любой части программы MyClass(); // конструктор ~MyClass(); // деструктор protected: // элементы в этой секции доступны из класса и его потомков private: // элементы в этой секции доступны только из класса; это область доступа по умолчанию }; константы (constants), представленные ключевым словом const,— это поля, остающиеся постоянными в течение всего времени жизни приложения. Описываются в полях класса public const int g = 10;

15. Область видимости класса: оператор разрешения области видимости ( : : ) - унарная форма, бинарная форма; вложенные классы - классы, вложенные в классы, классы, вложенные в блоки, вложенные определения функций.

Локальная область видимости – это область видимости внутри блока. Каждая функция – это отдельная область видимости. Внутри функции может быть несколько блоков, заключенных в фигурные скобки {...}, также образующих отдельные области видимости. Унарная форма записи: :: <идентификатор> Унарная форма используется для обращения к имени, относящемуся ко внешнему контексту и скрытому локальным контекстом или контекстом класса. Бинарная форма записи: <имя\_класса> :: <идентификатор> Бинарная форма используется для ссылки на контекст класса с целью устранения неоднозначности имен, которые могут повторно использоваться внутри класса. class cl1 {void f() {...} ...}; cl1 :: f(); // обращение к f() из cl1 Использование вложенных классов - это ограниченная форма вложенности функций. Функции должны определяться внутри локального класса и на них нельзя ссылаться внутри этого контекста. Вложенный класс не находится в области действия включающего его класса.

16. Компонентные данные: назначение, основные свойства, синтаксис и семантика определений компонентных данных, статические компонентные данные, статические константы, изменяемые компонентные данные (mutable).

Компонентными данными служат переменные, массивы, указатели и т.д. объявленные а базовых и производных классах, т.е используются для описания сущности класса. class point { float х, у, z; long а, b, с; }; static –статические поля класса . Память под такие поля резервируется при запуске программы, то есть еще до того, как программист явно создаст первый объект данного абстрактного типа. При этом все объекты, сколько бы их ни было, используют эту заранее созданную одну - единственную копию своего статического члена. Статический член класса должен быть инициализирован после определения класса и до первого описания объекта этого класса. Статические методы выполняться, даже если экземпляр класса не существует. Без ключевого слова static, константа была бы обычным полем экземпляра класса. Ключевое слово mutable используется вместе с членами класса в качестве дополнительного модификатора типа. Оно указывает, что член класса является изменяемым, и его можно изменять в функциях, у которых указан модификатор const, а также у константных объектов.

17. Компонентные функции: назначение, основные свойства, доступ к компонентным функциям, доступ компонентных функций к компонентным данным - функции селекторы, функции модификаторы.

Компонентная функция должна быть обязательно описана в теле класса. В отличие от обычных (глобальных) функций компонентная функция имеет доступ ко всем компонентам класса. Функция–компонент класса имеет ту же область видимости, что и класс, к которому она относится. Предназначены для работы с объектами класса. Для этих целей в определении функции указывается служебное слово (спецификатор) inline. <имя класса> : : <имя компонентной функции> Если определение принадлежит классу функции полностью размещено в классе то эта функция по умолчанию считается подставляемой. Подставляемая функция не может быть рекурсивной, не может содержать циклы, переключатели и т.д. Внутри тела класса помещается только прототип компонентной функции, а её определение – вне класса, как определение любой другой функции, входящей в программу. При таком внешнем определении компонентной функции она также может быть снабжена спецификатором inline При внешнем определении компонентной функции в теле класса помещается её прототип: <тип> <имя функции> (<спецификация и инициализация параметров>); Вне тела класса компонентная функция определяется таким образом: <тип> <имя функции> (<спецификация формальных параметров>) { <тело, принадлежащее классу функции> } Модификаторы позволяют установить значение private переменной-члена. Обычно имя модификатора состоит из Set\_ и имени переменной-члена. // установка показателя преломления void Set\_n(double n); Селекторы позволяют узнать значение private переменной-члена. // получение показателя преломления double Get\_n() const;

18. Встроенные компонентные функции: назначение, синтаксис и семантика определения, рекомендации по использованию; компонентные функции, определяемые вне класса: назначение, синтаксис и семантика определения и объявления, рекомендации по использованию Вне тела класса компонентная функция определяется таким образом: тип имя класса:: имя\_функции (спецификация\_формальных\_параметров) {тело\_принадлежащей\_классу\_функции} Описание класса с внешним определением его компонентных функций дает возможность, не меняя интерфейс объектов класса с другими частями программы, по-разному определять его компонентные функции.

19. Указатель this, компонентные функции типа static и const (статические и постоянные компонентные функции): назначение, синтаксис и семантика определения и объявления, рекомендации по использованию.

Указатель this используется в тех случаях, когда в теле принадлежащей классу функции нужно явно задать адрес того объекта, для которого она вызвана. Когда указатель this использован в функции, принадлежащей классу, например, с именем ZOB, то он имеет по умолчанию тип ZOB \*const и всегда равен адресу того объекта, для которого вызвана компонентная функция. Примером широко распространенного использования this являются операции со связанными списками. Чтобы компонент класса был в единственном экземпляре и не тиражировался при создании каждого нового объекта класса, он должен быть определен в классе как статический, т.е. должен иметь атрибут static. static int k; // объявление static-члена в объявлении класса - статические компоненты-функции не имеют указателя; - не могут быть объявлены две одинаковые функции с одинаковыми именами и типами аргументов; - статические компоненты-функции не могут быть виртуальными.

20. Конструкторы: назначение, наличие нескольких конструкторов у одного класса - причины, разделение функций, способы задания конструктора: по умолчанию или явное.

В C++ для упрощения процесса инициализации объекта предусмотрена специальная функция, которая называется конструктором. Имя конструктора совпадает с именем класса, конструктор запускается автоматически при объявлении переменной типа class. Часто бывает удобно указать несколько способов инициализации объекта. Для этого нужно описать несколько конструкторов Конструктор класса вызывается ав­томатически при объявлении объекта класса (по умолчанию). Если конструктор вызывается при объявлении объекта, то созданный в результате вызова конструктора объект фактически является объявляемым объектом. Единственное синтаксическое ограничение, налагаемое на конструктор, состоит в том, что он не должен иметь тип возвращаемого значения, даже void.

21. Явное описание (определение или объявление) конструктора, размещение определения в теле класса и влияние размещения на функции конструктора, размещение определения вне своего класса, параметры конструктора и возможности их использования.

Конструктор по умолчанию class myclass{};int main(){myclass mc(); } Конструктор с параметрами class Point { int x, y; Point(int x, int y) { this.x = x; this.y = y; } } Формальные параметры конструктора идентичны в структуре и поведении к формальным параметрам метода. Формальные параметры определяются списком разделенных от запятой спецификаторов параметра. Каждый спецификатор параметра состоит из типа и идентификатор (дополнительно сопровождаемый скобками), который определяет имя параметра.

22. Конструкторы для инициализации объектов, инициализаторы конструктора; конструкторы для преобразования типов, ключевое слово explicit.

Объект класса Account, инициализируемый конструктором, можно объявить следующим образом: Имя конструктора затем список обектов int main(){ // правильно: в обоих случаях вызывается конструктор // с двумя параметрами Account acct( "Ethan Stern" );//Конструктор инициализации обектов Account \*pact = new Account( "Michael Lieberman", 5000 ); Конструкторы в C++ (﻿﻿ctors), имеющие только один параметр, автоматически выполняют неявное преобразование типов. Например, если передать int, когда конструктор ожидает параметр указателя на строку, компилятор добавит код, необходимый для преобразования int в указатель на строку. Ключевое слово explicit применяется для указания явного преобразования типов. Смотрим пример. using System; class MyClass{ public MyClass(int i) { x=i; } public static explicit operator MyClass(int i) { return new MyClass(i); } private int x; оператор преобразования типа MyClass, который преобразует int

23. Конструкторы умолчания: назначение, способы задания, правила описания.

Конструктор по умолчанию — конструктор, который может быть вызван без аргументов. Если программист не определил конструктор в классе - то компилятор создаст конструктор (без параметров) для создания объектов этого класса. Если производный класс не вызывает явным образом конструктор базового класса, то конструктор по умолчанию вызывается неявно. class myclass{};int main(){myclass mc(); }

24. Конструкторы копирования: назначение, способы задания, правила описания.

Конструктор копирования Еще одним видом конструктора является конструктор копирования, который позволяет создавать копию экземпляра класса. Это актуально тогда, когда необходимы два экземпляра класса с одними и теми же значениями членов класса. Синтаксис заголовка конструктора копирования следующий: public: name\_constructor(type\_class & name); Обычно компилятор автоматически создает конструктор копирования для каждого класса, но в некоторых случаях программист создает конструктор копирования, называемый в таком случае явным конструктором копирования. Существует четыре случая вызова конструктора копирования: § Когда объект возвращает значение § Когда объект передается (функции) по значению в качестве аргумента § Когда объект конструируется на основе другого объекта (того же класса) § Когда компилятор генерирует временный объект

25. Вызов конструктора: назначение, способы реализации, задание аргументов; явный и автоматический вызовы: ситуации использования, рекомендации по применению.

Автоматический вызов происходит только для конструкторов по умолчанию. Явный вызов Сначала вызывается конструктор базового класса, затем инициализируются члены базового класса в порядке их появления в объявлении класса. После этого вызывается конструктор производного класса. В случае явного объявления конструкторов компилятор не предоставляет конструктор по умолчанию. Конструктор по умолчанию: Конструктор может иметь аргументы по умолчанию Class X (int=0) X (int j=1; float k=2); При вызове конструктора следует избегать неоднозначностей. X mx(1,2); Конструктор копирования: Для класса Х это такой конструктор, который может быть вызван с единственным аргументом - ссылкой на Х. X:: X(const &X)

26. Деструкторы: назначение, основные свойства, способы задания: по умолчанию или явное.

Они вызываются при уничтожении (отмене выделения памяти) объектов. Для того чтобы указать функцию как деструктор класса, необходимо указать перед именем класса знак тильды (~). По умолчанию деструктор задаётся ~String()

27. Явное описание (определение или объявление) деструктора, размещение определения в теле класса и влияние размещения на функции деструктора, размещение определения вне своего класса, параметры деструктора и возможности их использования.

class String {public: String( char \*ch ); // Declare constructor ~String(); String::~String() { if (\_text) delete[] \_text; В приведенном выше примере в деструкторе String::~String используется оператор delete, который отменяет выделение пространства, динамически выделенного для хранения текста.

28. Вызов деструктора: назначение, способы реализации, явный и автоматический вызовы: ситуации использования, рекомендации по применению.

Вызов деструктора для объекта абстрактного типа производится автоматически при его выходе из области существования. Для локальных переменных деструктор вызывается при выходе из блока, в котором эта переменная была объявлена. Для глобальных переменных вызов деструктора является частью процедуры завершения программы, выполняемой после функции main(). class cl{int num; public: cl (int i){num = i;} ~cl (){} }; void main (){ // Создание объекта obj типа cl: cl obj (1); : cl \*ptr = &cl (2); { cl \*tmp = ptr; // вызов дестр } } Вызов деструктора можно осуществить явно по его полному имени class cl{int num; public: cl (int i){num = i;} ~cl(){} }; void main(){cl obj (1); cl \*ptr = &cl (2); obj.cl::~cl (); ptr -> cl::~cl (); }

29. Специальный полиморфизм – преобразование АТД к встроенному типу.

Преобразования типов при присваивании В операциях присваивания тип значения, которое присваивается, преобразуется к типу переменной, получающей это значение. Явные преобразования типов Явное преобразование типа может быть выполнено посредством операции приведения типа. Она имеет следующую синтаксическую форму (<абстрактное-имя-типа>) <операнд> Преобразование операнда осуществляется так, как если бы он присваивался переменной типа <имя-типа> Результирующее значение типа void Преобразования типов при вызовах функций Метод преобразования аргументов функция при ее вызове зависит от того, имеется ли предварительное объявление данной функции, содержащее список типов ее аргументов. Если предварительное объявление имеется, и оно содержит список типов аргументов, то компилятор осуществляет контроль типов. Если предварительное объявление отсутствует, или в нем опущен список типов аргументов, то над аргументами вызываемой функции выполняются только преобразования по умолчанию. Преобразования выполняются отдельно для каждого аргумента вызова.

30. Специальный полиморфизм – перегрузка функций, алгоритм выбора перегруженной функции.

Перегруженные функции – это функции с одним и тем же именем, но имеющие разные списки параметров. Параметры могут отличаться типами и/или количеством. Алгоритм выбора перегруженной функции 1. строгое соответствие 2. стандартное повышение типа. 3. стандартное преобразование типа. Стандартное повышение типа лучше, чем остальные стандартные преобразования. Повышение - это преобразование float в double, а также bool, char в int. Кроме того, к стандартным преобразованиям относятся преобразования массивов похожих целых типов.

31. Дружественные функции: основные свойства, номенклатура, необходимость, полезность и опасность применения, наборы дружественных функций, дружественные классы.

Дружественной функцией класса называется функция, которая, не являясь его компонентом, имеет доступ к его собственным (private) и защищенным (protected) компонентам. Для получения прав друга функция должна быть описана в теле класса со спецификатором friend. Свойства: 1. Дружественная функция при вызове не получает указателя this. 2. Объекты классов должны передаваться дружественной функции только явно через аппарат параметров. 3. Дружественные функции нельзя вызывать через объекты классов, друзьями которых они являются, а также через указатели на эти объекты. 4. На дружественную функцию не распространяется действие спецификаторов доступа (public, protected, private) 5. Дружественная функция не может быть компонентной функцией того класса, по отношению к которому определяется как дружественная, зато она может быть просто глобальной функцией, а также компонентной функцией другого ранее определенного класса. 6. Дружественная функция может быть дружественной по отношению к нескольким классам. Дружественные классы Класс тоже может быть дружественным другому классу. Особенности: 1. Дружественный класс должен быть определен вне тела класса, "предоставляющего дружбу". 2. Все компонентные функции класса-друга будут являться дружественными для другого класса без указания спецификатора friend. 3. Все компоненты класса доступны в дружественном классе, но не наоборот.

32. Дружественные функции: объявление – размещение, синтаксис, семантика, требования к параметрам и типу возвращаемого значения; определение – размещение, область видимости; вызов функции.

Размещение друж.функций class Woman25 { private: char \*name;//имя int weight;//вес friend void setData(char \*, int, Woman25&);//объявление дружественных функций с парметрами friend void getData(Woman25&); тип возвращаемого значения (объекты, ссылки, указатели) перед типом возвращаемого функцией значения используется зарезервированное слово friend ... }; определение вне класса дружественных функций void setData(char \*n, int w, Woman25& object)//определяем friend-функцию setData { strcpy(object.name, n);//////////// object.weight = w; } void Woman25::setData(char \*n, int w)//определяем set-метод класса для дальнейшего вызова друж.функции в mаin() { strcpy(name, n); weight = w; } В друж функции не используется оператор :: (область видимости). Это говорит нам о том, что дружественная функция не принадлежит классу, не является его компонентом. А при определении методов использование оператора :: является обязательным.

33. Специальный полиморфизм – перегрузка операторов: синтаксис, семантика, приоритет и порядок выполнения, доступные и недоступные для перегрузки операторы, перегружаемые как обычные функции, только как компонентные функции.

Перегрузка оператора состоит в изменении смысла, при использовании его с определенным классом. Для того, чтобы перегрузить оператор, необходимо определить, что именно означает оператор по отношению к тому классу, к которому он применяется. тип имя\_класса::operator#(список\_аргументов) { // действия, определенные применительно к классу } Здесь перегруженный оператор подставляется вместо символа #, а тип задает тип значений, воз­вращаемых оператором. Для того, чтобы упростить использование перегруженного оператора в сложных выражениях, в качестве возвращаемого значения часто выбирают тот же самый тип, что и класс, для которого перегружается оператор. В общем случае ваши программы могут перегрузить почти все операторы С++. Операторы C++, которые программы не могут перегрузить. . Выбор элемента object.member .\* Указатель на элемент object.\*member :: Разрешение области видимости classname::member ?: Условный оператор сравнения с = (а > b) ? а : b Приоритет – формальное свойство оператора влияющеп на его выполнение в выражении. Например \* будет высшее приоритетом чем + или – Приоритет может быть равным, этот случай называется ассоциативностью. Любой унарный оператор определяется двумя способами. Либо как компонентная функция без параметров, либо как глобальная с одним параметром. Любой бинарный оператор определяется двумя способами. Либо как компонентная функция с одним параметром, либо как глобальная с двумя параметрами За исключением перегрузки оператором new и delete функция должна быть, либо не статичной компонентой функцией, либо либо иметь хотя бы один аргумент типа класс или ссыдку на класс. Операторы =, [],-> можно перегружать только с помощью нестатичной компонентной функции operator.

34. Перегрузка унарных и бинарных операторов: синтаксис и семантика.

Пример перегрузки бинарной операции + class three\_d { int x, y, z; // трехмерные координаты public: three\_d operator=(three\_d t); void show (); void assign (int mx, int my, int mz); }; // перегрузка + (например +, -, <<, >>) three\_d three\_d::operator+(three\_d t) { three\_d temp; temp.x = x+t.x; temp.у = y+t.y; temp.z = z+t.z; return temp; } Можно перегрузить унарные операторы, такие как ++ или --. При перегрузке унарного оператора с использованием функции, эта функция не имеет аргументов. Вместо этого операция выполняется над объектом, осуществляющим вызов функции-оператора путем неявной передачи указателя this. // перегрузка унарного оператора three\_d three\_d::operator++() { х++; у++; z++; return \*this; }

35. Перегрузка операторов присваивания и индексирования: синтаксис и семантика.

Оператор присваивания обязательно определяется в виде функции класса, потому что он неразрывно связан с объектом, находящимся слева от "=". Определение оператора присваивания в глобальном виде сделало бы возможным переопределение стандартного поведения оператора "=". class Integer { private: int value; public: Integer(int i): value(i) {} Integer& operator=(const Integer& right) { //проверка на самоприсваивание if (this == &right) { return \*this; } value = right.value; return \*this; } };

36. Перегрузка операторов ввода-вывода (<<, >>): синтаксис и семантика.

Операторы << >> в зависимости от контекста могут действовать как операторы ввода-вывода или как операторы сдвига. class MyClass { … private: freind std::istream & operator >>(std::istream & input, MyClass & var) { input >> var.getValue(); return input; } freind std::ostream & operator <<(std::ostream & output, MyClass const & var) { output << var.getValue() << endl; return output; } }

37. Перегрузка оператора () для индексирования: синтаксис и семантика.

оператор индексирования обычно перегружается, когда тип класса представляет агрегат . Теоретически, операция индексирования должна возвращать ссылку на элемент, который содержится в агрегате. double& operator [] (int); // Перегруженный оператор индексации double& Vector::operator [] (int n) { // Перегруженный оператор индексации // для проверки выхода за границы массива if(n < 0) // В случае если индекс меньше нуля, // то возвращаем нулевой элемент массива { cout << ""; return vect[0]; }

38. Перегрузка операторов доступа к элементу класса через указатель на объект ->: синтаксис и семантика

Указатель на объект объявляется с использованием того же синтаксиса, что и указатели на данные других типов. // простой пример использования указателя на объект #include <iostream.h> class P\_example { int num; public: void set\_num(int val) {num = val; } void show\_num(); }; void P\_example::show\_num() { cout << num << " \n"; } int main() { P\_example ob, \*p; // объявление объекта и указателя на него ob.set\_num(1); // прямой доступ к ob ob.show\_num(); р = &ob; // присвоение р адреса ob p->show\_num(); // доступ к ob с помощью указателя return 0; }

39. Перегрузка операторов new и delete: синтаксис и семантика.

Перегрузка операторов new и delete функция должна быть, либо не статичной компонентой функцией, либо иметь хотя бы один аргумент типа класс или ссылку на класс. Для перегрузки операторов new и delete может использоваться следующий формат: void \*operator new(size\_t размер) { // выполнение выделения return указатель\_на\_память; } void operator delete(void \*p) { // освобождение памяти, на которую указывает р } Функция delete получает указатель на область памяти, которую необходимо освободить. Для перегрузки операторов new и delete применительно к масси­вам: void \*operator new[](size\_t размер) { // выполнение выделения return указатель\_на\_память; } void operator delete[](void \*р) { // освобождение памяти, на которую указывает р }

40. Параметрический полиморфизм – шаблонные классы и шаблонные функции: назначение, параметризованные типы данных, синтаксис и семантика.

Классы-шаблоны полезны тогда, когда класс содержит логику, допускающую значительные обобщения. Используя классы- шаблоны, можно создавать классы, поддерживающие очереди, связанные списки и т. д. для про­извольных типов данных. Компилятор автоматически создаст корректный код, основываясь на типе данных, указанном перед компиляцией. Общая форма объявления класса-шаблона показа­на ниже: template < class птип> class имя\_класса { ... } Шаблоны функций­ – это предписание, по которому компилятор создает функцию для заданных типов данных. Шаблон функции имеет вид template < class тип, int size> тип имя\_функции ( параметры ) {//тело}

41. Шаблоны классов: назначение, синтаксис и семантика, объявление и определение шаблона, дружественные шаблонному классу функции, статические элементы шаблонного класса, аргументы шаблона класса.

Шаблоны классов­ – это предписание, по которому компилятор создаёт класс для заданных входных параметров.­ Шаблон функции начинается с ключевого слова template, за которым в угловых скобках следует список параметров. Затем следует объявление функции. В классе, реализующем связный список целых чисел, для работы с объектами. Для использования шаблона класса, необходимо указать его параметры. Параметрами шаблонов могут быть: параметры-типы, параметры обычных типов, параметры-шаблоны template< class T > class List { /\* ... \*/ public: void Add( const T& Element ); bool Find( const T& Element ); /\* ... \*/ }; Статические переменные не исчезают, когда содержащая их функция закончит свою работу, компилятор хранит их значения от одного вызова функции до другого. Инициализации статического поля выполняется вне класса программист должен использовать следующим образом: template <class T1> T1 имя\_класса <T1>::название\_поля=число;

42. Шаблоны функций: назначение, синтаксис и семантика, объявление, определение, вызов и перегрузка шаблонных функций, аргументы шаблона функции.

Шаблоны функций­ – это предписание, по которому компилятор создает функцию для заданных типов данных. Шаблон функции имеет вид template < class тип, int size> тип имя\_функции ( параметры ) {//тело} Все параметры шаблона обязательно должны использоваться при описании формальных параметров функции. template < class T1, class T2> void swap(int &af int &b)// перегрузка функции Вызов перегруженной шаблонной функции swap(i, j); При вызове функции swap (i, j) вызывается перегруженная явным образом версия swap()

43. Наследование и иерархия классов: основные элементы методологии объектно-ориентированного проектирования, реализация механизма повторного использования кода, полиморфная обработка родственных объектов – чистый полиморфизм.

Прежде всего это перегрузка методов классов при наследовании — традиционный для ООП способ обеспечения абстракции данных. Затем есть два способа реализации параметрического полиморфизма Первый способ— использование бестипового указателя и приведение типа в зависимости от других данных. Второй заключается в использовании шаблонов — но, в отличие от обычных реализаций параметрического полиморфизма, в С++ происходит автоматическая генерация семейства перегруженных функций на основании полиморфного определения (шаблона)