ООП

Лекция №1

Тема 1.2 основные принципы, методы, перспективы, разряды ООП.  
  
В окончательном виде любая программа представляет собой работу процессора. Все, что написано на любом языке программирования — это более упрощенная запись набора ... для процессора. Чем выше уровень языка ю, тем в более простой форме записываются одни и те же действия. В связи с колоссальным увеличением набора программ невозможно держать в голове все детали и становится необходимым отбрасывать несущественное. Это называется повышением степени абстракции программы. Для языка высокого уровня первым шагом к повышению абстракции является использование функций, позволяющих после написания функции отвлечься от деталей и реализации, поскольку для вызова функций необходимо знать только сигнатуру функции(интерфейс). Следующий шаг повышения абстракции — это описание собственного типа данных (пользовательских типов), позволяющий структурировать и обрабатывать информацию. Все это имеет единственную цель - упростить структуру программы, представить ее в виде небольшого количества блоков и оптимизировать взаимоотношения между ими. Это позволяет масштабировать приложения.  
  
ООП - методология программ, основанная на представлении все программы в виде совокупности объектов, каждый из которых является экземпляром конкретного класса, классы образуют иерархию.  
  
ЯП - поддерживаю некоторый стиль программирования, если в языке есть такие языковые конструкции, которые делают программу в этом стиле удобной, надежной, эффективной.  
  
В процедурном программировании упор делается на обработку данных с помощью алгоритмов.  
  
Язык С++ расширяет язык С средствами, которые поддерживают примирение ООП. Наиболее важными характеристиками ООП являются: абстракция, инкапсуляция и сокрытие данных, полиморфизм, наследование, повторное использование кода. Класс представляет собой наиболее важное расширение языка С++.  
  
Вся современная жизнь полна катастрофически сложных элементов. Единственный способ справиться это ограничить(?) упрощение абстракциями, которые позволяют представлять объект с тем объёмом информации, который необходим при решении определенной задачи.  
  
В функции объекта( в С++ методы) предназначается для доступа к данным класса, если необходимо получить некоторый данных объект, нужно вызвать метод, который выполнит эти действия. Прямой доступ к классам данных невозможен. Данные класса скрыты от внешнего воздействия, что защищает их от случайных или преднамеренных изменений (инкапсулированные в классе).  
  
Концепция чёрного ящика является одной из базовых в ООП. Снаружи объект принято рассматривать как чёрный ящик с кнопками и интерфейсном. Инкапсуляция позволяет изменить реализацию класса без модификаций основной части программы, если интерфейс остаётся прежним.  
  
Наследование (Inheritance) представляет собой возможность одного класса использовать методы другого без повторной его фактической реализации. Позволяет избавится от избыточности кода.

Лекция №2

Наследование представляет собой механизм, расширяющий существующие классы, сохраняя их функциональность и добавляя им новые свойства и методы. Класс, созданный в результате наследования другого класса наз. Производным, дочерним, подклассом (derived class, subclass). Класс, от которого дочерний наследуется наз. Предком, базовым, суперклассом. Наследование является механизмом повторного использования кода. «Reuse code». Наследование способствует независимому расширению программного кода. Установка отношения наследования между классами рождает иерархию.

При множественном наследовании (multiple inheritance) у класса может быть больше одного предка, в этом случае класс наследует все методы базовых классов. Множественное наследование реализовано на языке С++, но отсутствует на java и с#, которые могут возникнуть из-за наличия одинаковых имен в классах предков. Java и с# отказались от этого в пользу множественного наследования интерфейсов. В языке С++ для разрешения конфликтов использование наследованных методов с одинаковыми именами необходимо применять операцию наивысшего приоритета, которая называется оператором расширения образа видимости (оператор разрешения) «Scope Resolution Operator (::)».

Полиморфизм (polymorphism) – свойство, которое определяет одно и тоже имя, созданное в программе использовать для решения двух или более похожих, или, но технически разных реализаций решения задач. Целью полиморфизма в ООП является использование одного имени метода для создания общих действий, на выполнение каждого действия будет определятся типом данных, для которого это действие применяется. Перегрузка функций (function overloading). Главный принцип полиморфизма – один интерфейс и множество реализаций. Полиморфизм бывает двух видов: пол. Этапом выполнения и пол. Этапом компиляции. В пол. Этапа компиляции относятся: перегрузка функций и перегрузка операторов. К пол. Этапа выполнения (runtime) относится использование виртуальных функций.

Класс представляет собой чертеж, шаблон, остов, скелет, лекало. Объект класса всегда создается согласно начинке, которая определена в классе. Класс — это абстракция, которая описывает методы и свойства несуществующих объектов. Объекты класса (экземпляры) могут иметь различное поведение, свойства, но все равно будут является объектами одного класса. При создании объекта класса создается отдельная копия данных для каждого объекта этого класса, но методы создаются и помещаются в память только один раз. Все объекты класса используют одну реализацию методов этого класса.

Сокрытие данных не только предотвращает прямой доступ к данным, но и избавляет от необходимости знать как представлены данные. Поскольку одним из главных принципов ООП является сокрытие данных, то элементы и данные обычно размещаются в разделе private. Методы, которые образуют интерфейс класса размещаются обычно в разделе public. Закрытые методы применяются для управления деталями реализации класса, которые не формируют часть публичного интерфейса.

Как классы, так и структуры являются конструкциями, с помощью которых разработчик определяет собственные пользовательские типы. Единственная разница между структурами и классами состоит в том, что типом доступа по умолчанию у структур является public. В большинстве случаев программисты используют классы для определения объектов, содержащих и данные, и поведение. Структуры же используются для определения объектов, содержащих данные и не требующих описания поведения.

Реализация метода класса должна иметь тип возвращаемого значения, имя класса, оператор разрешения контекста, и набор методов, которые есть у класса. Методы класса имеют доступ к private компонентам класса.

**Конструктор класса**

Объявление классов, включая его методы, образуют «сервер», который является ресурсом доступным, нуждающимся в нем, программам. Клиент взаимодействует с классом только через открытый интерфейс. Единственной ответственностью клиента (программиста) является знание интерфейса и его правильное использование.

Одна из целей языка С++ состоит в том, чтобы сделать использование объектов классов подобным применению стандартных типов данных. Единственным способом корректного воссоздания(валидного) объекта является автоматическая инициализация объектов при их создании. Конструкторы класса предназначены для создания новых объектов и присваивания им корректных значений. Имя конструктора всегда совпадает с именем класса. Конструктор не имеет возвращаемого значения и никогда не объявляется даже с типом void. Конструктор вызывается автоматически при создании объекта. Конструкторы можно перегружать.

Лекция №3

Язык С++ предлагает 2 способа инициализации объектов с помощью конструктора: явно и неявно.

Stock stock1 = stock(“name”, 250, 1.25) – явно

Stock stock2= Stock(“name”, 50,2.25)

Конструктор вызывается всякий раз, когда создается объекта даже есть применяется … выделения памяти. Конструктор не вызывается объектом, а служит для его создания.

**Конструктор по умолчанию**

Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации объектов. Это конструктор, который используется для создания объекта, когда не указаны какие-либо явные инициализирующие значения. Если вы не создадите ни одного конструктора класса, то компилятор с++ автоматическисоздаст конструктор по умолчанию. Компилятор не создаст конструктор по умолчанию если в классе есть хотя бы один конструктора с параметрами.

Stock::Stock()

Конструктор по умолчанию можно создать 2 способами один из них предусматривает указание значения по умолчанию для всех аргументов. Допускается наличие только одного конструктора по умолчанию.

**Деструкторы класса**

В момент разрушения объекта программа автоматически вызывает специальную функцию «элемент», которая называется деструктор. Он предназначен для того, чтобы очистить те участки памяти, которые были использованы при создании объекта. Как и конструктор деструктор имеет специальное имя, которое состоит из имени класса и значка(символа) тильда. Деструктор никогда не имеет возвращаемого значения, и никогда не имеет аргумента. Деструктор в классе всегда будет один и моментом, когда он вызывается управляет ваша исполняющая среда.

**Указатель “this”**

Каждый объект класса содержит свою копию полей, а методы не дублируются. Каждый не статический метод класса, помимо явно объявленных параметров, получает еще один крытый параметр**.** Это константный указатель на объект, для которого он вызван. Указатель this всегда указывает на начало объекта в ОП. Следовательно, в любой нестатической функции можно использовать указатель this для обращения полей текущего объекта.

Функция друзья (friend функция) не имеют указателя this поскольку они не являются элементами класса.

**Массив объектов пользовательского типа**

Массивы объектов какого-либо класса объявляются таким же способом, как и массивы стандартных встроенных типов.

Тема 2.1 Инкапсуляция и организация доступа к элементам объекта.

Встраиваемым, называется метод, который вставляется в то место программы, из которого он вызывается. Существует 2 способа создания встраиваемых методов: первый состоит в использовании модификатора inline, второй в реализации методов внутри класса. Модификатор inline должен предворять все остальные аспекты явления методов. Встраивать необходимо небольшой по объему метод. Причиной существования встроенных методов является эффективность их использования.

При каждом вызове обычного метода должна быть выполнена целая последовательность инструкций, связанных с обработкой самого вызова, включающего помещение его аргумента в стек. Значительное количество циклов центрального процессора используется для выполнения именно этих действий. Использование модификатора inline является запросом... Существует ряд ограничений, которые не позволяют компилятору выполнить такой запрос. Например, если метод содержит цикл, switch, go to, рекурсию, либо статические переменные. Ограничения на использование встроенных методов зависит от конкретной реализации системы.

**Указатели на объекты**

Доступ к содержимому объекта можно получить либо через сам объект, либо через указатель. Для доступа к полю объекта через указатель используется оператор «->». При инкрементации или декрементации, чтобы он всегда указывал на следующий или предыдущий элемент базового типа. Указатель на объекты классов используются при реализации динамического полиморфизма.

**Функции Друзья**

В С++ существует возможность разрешить доступ закрытым элементам класса, методам, которые не являются элементами этого класса. Для этого необходимо определить методы дружественными по отношению к рассматриваемому классу. Чтобы сделать функцию другом класса необходимо прототип ключевым словом friend. Один метод может быть другом нескольких классов. Одна из причин использования friend функций связаны с тем, что 2 класса использую одну и туже функцию.

Лекция №4

Friend-функция вызывается обычным образом, то есть без привязки к объекту. Обычно в friend функции в качестве параметра передаются один или несколько объектов классов, для которого функция является friend-ом. Friend функции будут необходимы при реализации перегрузки операторов.

Опережающее объявление класса предназначено для объявления имени типа до определения самого класса. Чтобы создать опережающее определение класса необходимо написать ключевое слово class, имя класса и поставить «;». Друг одного класса может быть элементом другого класса.

**Перегрузка функций**

Перегрузка функций — это механизм, который позволяет двум родственным по своим задачам функциям иметь одинаковые имена. Несколько функций могут иметь одинаковые имена при условии, что их параметры различаются. Функции считаются различными, если они отличаются по количеству, типу и порядку следования аргумента. Перегрузка функций – это один из способов реализации полиморфизма этапа компиляции. Поскольку списки параметров для всех версий перегруженных функций различны, то компилятор обладает достаточной информацией, чтобы вызвать правильную версию каждой функции. Для определения того, какую версию перегруженной функции вызвать компилятор использует количество типов и порядок следования параметров(аргументов), тип возвращаемого значения недостаточен для реализации перегрузки. Каждая версия перегруженной функции может выполнять абсолютно любые действия, но сточки зрения здравого смысла перегрузка функций подразумевает родство ее версий.

**Перегрузка конструкторов**

Для перегрузки конструктора класса достаточно объявить его во всех нужных сигнатурах и определить каждое действие, связанное с каждой действующей сигнатурой.

Тип clock\_t представляет собой псевдоним длинного целочисленного типа способного представлять тактовые счетчики. Тактовые счетчики — это единицы времени постоянной, но системно-зависимой величины возвращаемые функциональными часами.

При разработки полноценной библиотеки класса стоит предусмотреть набор различных конструкторов, охватывающий тот спектр форматов инициализации, который обеспечивает наиболее подходящие способы создания объектов.

**Динамическая и статическая инициализация переменных и объектов**

Как локальные, так и глобальные переменные можно инициализировать в момент выполнения программы. Процесс инициализации в момент выполнения программы называется динамической инициализацией. Во время динамической инициализации можно использовать любое действительное значение соответствующего типа. Наличие множества форматов инициализации избавляет программиста от выполнения дополнительных преобразований при инициализации объекта.

Загромождение кода конструкторами для обработки едко возникающих ситуаций оказывает дестабилизирующее влияние на программу.

**Присваивание объектов**

Если два объекта имеют одинаковый тип, то один объект можно присвоить другому. Для присваивания недостаточно, чтобы два класса были физически подобны. Имена классов, которые участвуют в присваивании должны совпадать. Если один объект присваивается другому, то по умолчанию данные первого объекта поразрядно копируются во второй объект.

Лекция №5

Присваивание одного объекта другому, делает эти объекты идентичными, но совершенно независимыми. В С++ объекты передаются функция путем использования соглашения о передаче параметров по значению. При этом функции передается не сам объект, а его копия.

Несмотря на то, что передача функциям не сложных объектов в качестве аргументов простая процедура, при этом могут происходить непредвиденные события имеющие отношения к конструкторам и деструкторам. При передаче объекта какому-нибудь методу, создается его копия и эта копия становится параметром метода. Создание копии означает конструирование нового объекта, когда выполнение метода завершается, копия аргумента (параметр метода) разрушается. Когда при вызове метода создается копия аргумента, обычный конструктор не вызывается. Вместо этого вызывается конструктор копий. Конструктор копий определяет как должны быть создана копия объекта, но если в классе явно не определен конструктор копий, то С++ предоставит его по умолчанию. Конструктор копий по умолчанию создаёт побитовую (поразрядную, идентичную) копию объекта. Поскольку обычный конструктор используется для инициализации параметров нового объекта, он не должен вызываться для создания копий. При передаче объекта методу имеет смысл использовать текущее состояние объекта, а не его изначальной. Когда метод завершается и разрушается копия объекта, используемая в качестве аргумента метода, и вызывается деструктор этого объекта. Необходимость вызова деструктора связана с выходом объекта из области видимости.

**Потенциальные проблемы в передаче объектов в качестве параметра**

Несмотря на то, что объекты в С++ передаются функциям по значению, что теоретически защищает функцию? изменения, здесь возможен побочный эффект при работе с динамической памятью. Если объект, используемый как аргумент, требует динамического выделения памяти и освобождает эту память при разрушении, его локальная копия при вызове деструктора освободит ту же самую область памяти, которая была выделена оригинальному объекту. Это становится потенциальной проблемой, поскольку оригинальный объект все еще используют эту, уже освобожденную, область памяти.

Работа конструктора копирования заключается в том, чтобы создать реальную копию объекта со своей личной выделенной динамической памятью.

Ошибки времени выполнения (runtime error) самые плохие ошибки, которые могут возникнуть в вашей программе. Повторное освобождение памяти считается определенной операцией, которая как правило в зависимости от реализации системы динамического распределения памяти вызывает н.. ошибку. Один из путей решения этой проблемы состоит в передаче объекта не по значению, а по ссылке или через указатель. В этом случае копии объекта не создаются, а деструктор не вызывается. Для того, чтобы функция могла вернуть объект, необходимо объявить в качестве типа возвращаемого ей значения тип соответствующего класса. При этом, если метод возвращает объект класса, то он автоматически создает временный объект, в котором хранится возвращаемое значение. После возврата значения из метода объект разрушается.

**Конструктор копий**

Имя\_класса (const имя\_класса &obj){

//тело

}

Конструктор копий вызывается при создании объекта только в случае его одновременной инициализации содержимым другого объекта.

Практически каждый класс, профессионально описанный в программах, содержит явно определенный конструктор копий. Позволяющий избежать побочных эффектов побитового копирования.

В Java и С# программисты не выделяют самостоятельно динамическую память, они работают с динамической памятью исключительно через ссылки.

Лекция №6

**Тема 3.1 Структурные элементы класса, методы взаимодействия объектов классов**

Переменные, объявленные внутри функции или блока, называются локальными. Глобальными называются переменные доступные для всех (объявлены вне какой-либо функции, их область видимости распространяется на всю программу, после места их объявления). Переменные типа static это переменные «долговременного» хранения, они хранят свои значения в пределах функции ил файла. От глобальных переменных статические переменные отличаются тем, что за рамками функции или файла они не известны. Локальные статические переменные сохраняют свое значение между вызовами функции. Если к локальным переменным применен спецификатор static, то для этой переменной выделяется область памяти. Это позволяет сохранять значение переменной между вызовами функций. Локальные статические переменные инициализируются только один раз при первом выполнении функций.

Глобальные статические переменные известны только файлу, в котором они объявлены. Это означает, хотя переменная является глобальной, другие функции не имеет о ней никакого понятия, соответственно не могу изменить ее значение. Несмотря на то, что глобальные статические переменные допустимы их использовать не стоит. Для управления доступа к глобальным переменным рекомендуется использовать пространство имен.

Пространство имен (namespace) позволяет вам именовать классы, функции, переменные короткими одинаковыми именами, но разрешать конфликт имен за счет имени пространства имен. Пространство имен объявляет некоторую область видимости. До появления ключевого слова namespace была определена в одном глобальном пространстве имен. Сейчас библиотека С++ определяется в собственном пространстве имен именуемом std.

**Статические компоненты класса**

Каждый объект класса содержит собственные данные. Если поле класса описано ключевым словом static, то значение этого поля будет одинаковых для все объектов данного класса. Статические поля класса используется в тех случаях, когда необходимо, чтобы все объекты включали в себя какое-то одинаковое значение. В отличии от обычных переменных элементов класса статические переменные элементы класса не копируются для каждого объекта отдельно. Статическое поле класса по своим характеристикам. Статическое поле класса по своим характеристикам похоже со статической переменной: оно видимо только внутри класса, но время его жизни совпадает со временем жизни программы. Так как статическое поле существует в одном экземпляре, даже если не созданы ни одного объекта класса, то к статическому полю можно обратится через имя класса. Статические элементы класса инициализируются нулем еще до создания первого объекта класса.

Статические переменные могут быть: открытыми, закрытыми и защищенными. Объявление статической переменной в классе не означает ее инициализацию. То есть память для статической переменной не выделяется, если переменная только объявлена в классе. Чтобы разместить статическую переменную в памяти, следует разместить ее вне класса, то есть глобально. Доступ к открытым статическим элементам класса возможен посредствам любого объекта этого класса или с помощью бинарной операции разрешения области видимости.

В некоторых объектно-ориентированных языках (Java) не существует глобальных переменных. Поэтому статические поля классов единственных способ хранения глобальных данных как общих ресурсов. Определение статических полей класса происходит не так, как для обычных (автоматических) полей класса. Обычные поля объявляются (компилятору сообщается имя и тип) и определяются (компилятор выделяет память для хранения поля) в одном операторе. Для статических полей объявление и определение переменной выполняется двумя разными операторами: объявление поля находится внутри определения класса, а определение поля располагается вне класса… . Метод класса может быть объявлен статически, если он обращается к статическим элементам класса. То есть статические методы класса могут обращаться непосредственно только к статическим полям и вызывать только другие статические методы класса, потому что им не передается указатель this. Статические методы не могут быть константными, виртуальными и не могут быть объявлены с ключевым словом volatile. Один и тот же метод не может иметь статическую и не статическую версию. Обращение к статическим методам может быть реализовано либо через имя объекта, либо через имя класса.

Ключевое слово const предшествует описанию типа переменной и означает, что переменная может быть проинициализирована лишь однажды и значение этой переменной не может быть изменено. Так же префикс const может использоваться к: аргументам функции и возвращаемым значениям, указателям, объектом классов, методам и полям классов.

**Константные аргументы функций**

Ссылочный механизм передачи аргумента в функцию позволяет функции изменять значение аргументов, но существуют и другие причины использования ссылок на аргументы функции, одно из них является эффективность. Так как некоторые переменные, передаваемые в качестве аргумента функции, могут иметь большой размер занимаемой памяти. Если аргумент занимает много места, то передача по ссылке является более эффективной и экономичной, поскольку при передаче по ссылке передается не значение переменной, а адрес. Если ссылочный механизм используется только из соображений эффективности и при этом необходимо запретить изменение аргумента, то необходимо указать идентификатор const перед соответствующим аргументом. Одним из положений, которых должен придерживаться хороших разработчик является в том, что «лучше обнаружить ошибку на этапе компиляции, чем на этапе выполнения».

Лекция №7

Константные методы не изменяют полей своего класса. Для того, чтобы сделать функцию константной, необходимо указать ключевое слово const после прототипа функции, но до начала тела функции. Если описание прототипа и реализация функции разделены, то модификатор const необходимо указывать дважды. Те методы, которые лишь считывают данные из полей класса, стоит сделать константными. Использование константных методов помогает компилятору, обнаружить ошибки на этапе компиляции, запрещая модификацию полей объекта. Если объект класса объявлен с модификатором const, то он становится недоступным для изменения. Это значит, что для него можно вызвать только константные методы, поскольку они гарантируют, что объект не будет изменен. При создании класса, стоит объявлять константными те методы, которые гарантированно не будут изменять содержимое текущего объекта.

**Вложенные классы**

Один класс можно объявить внутри другого, таким образом создаются вложенные классы. Вложенный класс является элементом объемлющего класса и его определение может находится в любом из секций public, private, protected. Поскольку объявление класса фактически определяет его область видимости, то вложенные классы могут существовать только внутри области видимости внешнего класса. Имя вложенного класса не конфликтует с таким же именем, объявленным в другой области видимости. Для обращения ко вложенному классу из области отличной от объемлющего класса следует использовать полное имя класса. Вложенные классы могут непосредственно использовать имена полей, имена статических элементов и перечислений только из включающего их класса.

**Перегрузка операторов для пользовательских типов данных**

Перегрузка может превратить сложный и малопонятной код программы в интуитивно понятный и очевидный. Перегрузка операций говорит о том, что обычным операциям С++ таким как «+» «\*» в случае их применения, определенными пользователем типом данных назначается свой смысл выполнения. Компилятор языка С++ определяет, какую реализацию оператора вызвать в зависимости от типов операндов(контекста). Для перегрузки любой операции необходимо использовать ключевое слово «operator». Сначала пишут возвращаемый тип, затем ключевое слово operator, затем саму операцию, которую вы перегружаете, затем список аргументов заключенных в скобки. Такой синтаксис говорит о том, что если оператор принадлежит классу для которого реализована его перегрузка, то нужно вызвать перегруженный оператор этого типа. // void operator++() {++count;}

Компилятор может различать перегруженный оператор по количеству типа и порядку следования аргументов. Перегружаемые операции компилятор также различает по типу данных их операндов. Так как методы класса всегда имеют доступ к содержимому объекта класса, для которых они были вызваны, то для операции имеющих один операнд объект не нужен. Для того, чтобы иметь возможность использовать унарный оператор в выражениях необходимо определить тип возвращаемого значения.

Для оптимизации кодов перегрузки операторов можно использовать временные безымянные объекты, которые создаются с использованием конструктора с необходимыми параметрами, но не имеют имени.

**Перегрузка бинарных операций**

Существует правило, по которому реализуется перегрузка бинарных операторов. Объект, стоящий с левой стороны операции, вызывает функцию оператора. Объекту, стоящему справа от знака операции, должен быть передан перегружаемый оператор в качестве параметра. Перегружаемая операция должна вернуть значение типа класса. Перегруженные операции всегда требуют количество аргументов на один меньше, чем количество операндов, так как левым операндом по умолчанию является текущий объект. Это правило не распространяется для функций являющимися дружественными.

**Преобразование типов**

Если значение одного объекта присваивается другому объекту того же типа, то значение всех переменных объекта просто копируются в новый объект. Такое присваивание выполняется без дополнительных усилий со стороны разработчика если по обеим сторонам от знака равно используются переменные одинаковых типов. Существует 2 вида преобразования: явное (Explicit) и неявное (Implicit).

Использование оператора static\_cast указывает на явное преобразование, которое может привести к потере данных, но программист это осознает и допускает (допустимая потеря). Если нужно осуществить преобразование определенных пользователем типов в основные или встроенные, то программист должен написать функции для выполнения этих преобразований, так как никаких встроенных функций нет. При преобразовании от пользовательского типа к основному необходимо реализовать операцию наподобие

Operator float()

**Механизмы наследования и определение собственных типов данных**

Наследование – процесс создания новый производных классов (наследников), из уже существующих классов и добавление в дочерние классы новых полей или реализаций методов.

На диаграммах UMA наследование указывается стрелкой, которая называется генерализация (обобщение), которая указывает на базовый класс.

Лекция №8

Наследование позволяет использовать существующий код несколько раз. Имея написанный и налаженный с своей работе базовый класс, мы можем его больше не модифицировать, а наследование позволит приспособить базовый класс для работы в других ситуациях. Наследование может помочь также при начале постановки задачи при разработке общей структуры программы. Важным результатом повторного использования кода является упрощение распространения библиотек. Программист может использовать классы, созданные кем-то без модификации кода просто создавая производные классы. Для того, чтобы не тратить время на повторное тестирование логики базового класса, либо при невозможности модификации этого класса, мы можем использовать наследование для создания необходимых новых классов. Если поле объявлено в классе со спецификатором protected, то оно закрыто для использования его вне класса, но открыто для использования его дочерних классов.

Методы производного класса имеют доступ к элементам базового класса со спецификатором public and protected, а к полям с атрибутами private методы производного класса доступа не имеют.

**Недостатки использования спецификатора protected**

Допустим вы написали библиотеку классов и публично ее распространяете, любой программист может получить доступ к элементам классов, объявленных как protected, просто наследуя эти классы. Это делаем содержимое класса с атрибутом protected менее защищенным, чем private. При разработке больших программ вам придется сделать выбор между доступом к полям и их наследованию с использование атрибутов спецификатора protected. При наследовании базовый класс создается неизменным. Базовый класс ничего не знает о производных классах. Наследование не работает в обратном направлении. То есть базовому классу и объектам базового класса недоступны элементы производных классов.

**Конструкторы производного класса**

При создании объекта дочернего класса «без аргументов» компилятор будет использовать конструктор базового класса «без аргументов». Если необходимо создать объект дочернего класса с аргументами, то необходимо написать в классе соответствующий конструктор, который вызовет конструктор базового класса. При создании объектов дочернего класса с параметрами необходимо внутри тела конструктора вызвать конструктор базового класса. При конструировании экземпляра класса наследника всегда происходит предварительный вызов конструктора базового класса. Порядок вызова деструктора всегда кратен вызову конструктора: сначала вызывает деструктор класса наследника, затем деструктора всех базовых классов.

**Переопределение методов overriding**

Можно переопределить для производного класса метода, имеющего такие же имена, как и у метода базового класса. Это называется переопределением методов.

Переопределяемые методы имеют одинаковые имена, аргументы и возвращаемые значения. До того, чтобы поднять какой из двух методов (разового или производного) класса необходимо вызвать существует правило, если один и тот же метод существует и в базовом, и в производном классе, то для объектов производного класса будет выполнен метод из производного класса. В этом случае мы говорим, что метод производного класса переопределяет метод базового класса. Объектам базового класса ничего не известно о объектах производного класса, следовательно, для них всегда будут вызываться методы базового класса. Для получения доступа к методу базового класса из производного необходимо использовать операцию разрешения области видимости.

Inheritance should create an IS-A relationship. Meaning the child is a more specific version of the parent.

Язык С++ предоставляет несколько способов для точного регулирования доступа к элементам класса. Если при наследовании указывали ключевое слово public, то оно определяет, что объект производного класса может иметь доступ к полям метода базового класса, объявленные как public и protected. При использовании ключевого слова private для объектов производного класса нет доступа к методам базового класса, объявленным как public, private и protected. Если базовый класс наследуется как public класс, то public элементы базового класса становятся public элементами производного класса. Protected элементы базового класса…

Лекция №9

Если базовый класс наследуется как private, то public элементы становятся private элементами производного класса, а private поля недоступны в производных классах.

Если базовый класс наследуется как protected класс, то публичные и защищенные поля, становятся защищенными полями в защищенной классе, а private элементы недоступны в производном классе.

Если модификатор доступа при создании дочернего класса не указан, то по умолчанию будет использован private, а для структур public.

В большинстве случаев производный класс представляет собой специализированную версию базового класса, при такой реализации класса имеет смысл использовать публичным наследованием. Если необходимо в производном классе полностью модифицировать действия базового класса, скрывая или изменяя первоначальный вид, то стоит использовать private наследование. Использовать защищенное наследование можно, если внутри всех потомков и во всей иерархии наследования будут использоваться только protected элементы «передаваемые» наследованием, которые будут защищены от внешнего воздействия. Производные классы могут являться базовыми классами других производных классов.

**Множественное наследование**

Класс может быть производным не только от одного базового класса, а одновременноот двух и более *(это называется множественное наследование)*. Множественное наследование классов существует в С++, но отсутствует в языке java. В языке java присутствует множественное наследование интерфейсов.

**Конструкторы множественного наследования**

При множественном наследовании конструктор вызывается в порядке указания базовых классов. Деструкторы во множественном вызываются в таком порядке: с самого нижнего дочернего класса, затем конструктор класса, стоящий самым последним в строке классов, затем, которые перед ним и т.д.

Допустим, что в обоих базовых классах существуют методы с одинаковыми именами, а в производном классе такого метода нет. Возникает вопрос: как в таком случае объект производного класса определит какой из методов базового класса необходимо вызвать. Для разрешения этой неоднозначности можно использовать следующий синтаксис

Имя\_объекта.имя\_класса::имя метода

Если мы создаем производный класс от 2 базовых, которые в свою очередь, являются производными одного класса, то возникает наследование, которое называется ромбовидным.

При множественном наследовании невозможно однозначно определить какую реализацию метода из класса верхушки ромба необходимо вызвать.

**Связь включения: классы в классы**

Включение позволяет использовать классы в качестве типов полей других классов. Любой дочерний класс, наследуя свойства базового класса добавляет свои собственные поля или методы. Связь наследования всегда связана со словом «является». Включение показывает на отношения между классами типа «имеет или содержит». Например, библиотека включает книги или персонал. Включение называют взаимоотношения типа «имеет», «часть целого». Более сложным вариантом включения является композиция. Композиция обладает всеми свойствами включения, но при композиции, часть принадлежит целому и время жизни части то же самое, что время жизни целого.

Комната является частью квартиры, следовательно, здесь связь композиция, потому что комната без квартиры существовать не может. А мебель не является не является неотъемлемой частью квартиры, но в то же время, квартира содержит мебель, поэтому …

Лекция №10

Полиморфизм — принцип ООП, который описывает процесс, в котором различные реализации метода могут быть доступны по средствам одного и того же имени. «Один интерфейс и много реализаций» — это означает, что ко всем методам общего класса можно получить доступ одним и тем же способом, несмотря на возможные различия в конкретных действиях, связанных с каждой отдельной реализацией метода в дочернем классе. В языке С++ полиморфизм поддерживается как во время выполнения*(динамический)*, так и во время компиляции*(статический)*. Перегрузка операторов и функций — это полиморфизм этапа компиляции. Использование производных классов и виртуальных функций относятся к полиморфизму этапа выполнения. **Указатель на базовый класс может ссылаться на любой объект дочернего класса**.

Как правило указатели одного типа не могут указывать на объекты другого типа. Исключением из этого правила является указатели на базовые классы и объекты производного класса.

Через указатель на базовый класс невозможно получить доступ к тем элементам, которые специфичные для производного класса. Если через базовый класс нужно получить доступ к определенным элементам в производном классе, то нужно привести указатель базового класса к типу указателя на производный класс. Хотя базовый указатель можно использовать для доступа к объектам любого производного типа *(независимо от того, сколько уровней наследования было использовано)*, обратное утверждение не верно.

Указатели всегда инкрементируется и декрементируется относительно своего базового типа. Если указатель на базовый класс используется для доступа к объекту производного типа, то ++ или -- его не заставят ссылаться на следующий объект производного класса. Вместо этого он будет указывать на следующий объект базового класса.

**Ссылки на производные типы**

Подобно указателям, ссылку на базовый класс можно использовать для доступа к объекту производного типа. Параметр, который имеет тип ссылки на базовый класс, может принимать объекты базового класса, а также объект любого другого типа, выведенного из базового.

**Использование виртуальных методов**

Виртуальный метод — это метод, который объявляется в классе с использованием ключевого слова virtual и переопределяется в одном или нескольких производных классах. Каждый производный класс может иметь собственную версию виртуального метода. Предположим, что виртуальный метод вызывается через указатель или ссылку на базовый класс. В этом случае среда С++ определяет, какую именно версию виртуальной функции необходимо вызвать по типу объекта, на который ссылается указатель. Это решение принимается всегда во время выполнения программы.

Именно по типу адресуемого объекта, а не по типу самого указателя, определяется, какая версия виртуальной функции будет выполнена. При переопределении виртуальный виртуальной функции в производном классе ключевое слово virtual повторять не нужно. Класс, который включает в себя виртуальную функцию называется полиморфным.

**Только при обращении к виртуальной функции через указатель на базовый класс достигается динамический полиморфизм.**

Прототипы виртуальной функции и всех ее переопределений должны быть абсолютно одинаковыми. Виртуальная функция должны быть элементом класса, а не его другом. Деструкторы могут быть виртуальными, конструкторы — нет. Атрибут virtual передается по наследству. Если в производном классе не переопределяет виртуальную функцию, то для объектов производного класса будет вызываться реализация функции определенная базовым. Использование виртуальной функции позволяет базовому классу определять обобщенный интерфейс, который будет поддерживаться всем производными классами и при необходимости дочерних класса может быть изменен.

Лекция №11

Виртуальные функции в сочетании с производными типами позволяют поддерживать динамический полиморфизм, который дает возможность некоторому классу определить функции, которые будут использовать все производные классы, причем производный класс может определить собственную реализацию либо использовать реализацию базового класса. То есть базовый класс диктует общий интерфейс, который будет иметь любой объект, выведенный из этого класса, но позволяет при этом производному классу определить метод необходимый для реализации этого интерфейса.

Важность использования виртуальных функций опирается на тот факт, что при разработке программ можно вывести новый тип, который будет разделять общий интерфейс с другими родственными классами.

Чисто виртуальная функция *(pure virtual function)* – это виртуальная функция, которая не имеет определения в базовом классе. Виртуальная функция является элементом класса и может быть переопределена в дочерний. В то время как чисто виртуальная функция не имеет реализации в классе. Базовый класс имеет реализацию виртуальной функции, в то время как для чисто виртуальной функции реализации в базовом классе нет *(присутствует только ее прототип)*. Дочерние классы не обязаны переопределять виртуальную функцию, в то время как чисто виртуальную функцию дочерний класс должен переопределить, иначе он не может быть инстанциирован *(нельзя создать его объект).* Наличие в классе виртуальных функций не делает класс абстрактным, в то время как наличие хотя бы одной чисто виртуальной функции в классе делает класс абстрактным. Отсутствие переопределения виртуальной функции в дочернем классе не ведет к ошибке компиляции, в то время как отсутствие реализации чисто виртуальной функции в дочернем классе приводит к ошибке этапа компиляции.

Чисто виртуальные функции используются для того, чтобы гарантировать, что производный класс содержит реализации всех необходимых методов. *Чисто виртуальная функция используется для того, чтобы отложить реализацию функции.*

Абстрактный класс реализуется важной особенность: у такого класса не может быть главных объектов, но для такого класса можно создать ёёёёёёёёёёуказатель, который может ссылаться на любой объект любого дочернего класса.

При раннем связывании *(early binding)* вызов функции подготавливается во время компиляции, а при позднем связывании *(late binding)* – во время выполнения программы.

Тема 4 **ШАБЛОНЫ *(templates)***

Шаблоны позволяют достичь одну из самых трудных целей в разработке – создать многократно используемый код. Используя шаблоны можно создавать обобщенные функции и классы. По обобщенной функции и классе обрабатываемый ей *(им)* тип данных задается как параметр. Таким образом функцию или класс можно использовать для разных типов данных, не предоставляя явным образом конкретные версии для каждого типа данных. Обобщенная функция перегружает саму себя, определяя общий набор операторов, которые предназначены для применения к разным типам данных. Тип данных обрабатываемой функции передается как параметр. Многие алгоритмы имеют одинаковую логику данных. Например, алгоритм сортировки quick sort *(qsort)* применяется одинаково и к массиву целых чисел, и к массиву значений с плавающей точкой. Различие будет состоять только в типе сортируемых данных. Создавая обобщенную функцию, можно определить природу алгоритма независимо от типов данных, которые им обрабатываются. Обобщенная функция создается с помощью ключевого слова *template*, компилятору же остается дополнить недостающие детали в соответствии с заданным значением параметра типа шаблона.

Параметр *Ttype* представляет собой заполнитель для типа данных обрабатываемых функций. Это имя может быть использовано в теле функций, вместо которого компилятор автоматически подставит реальный тип данных при создании конкретной версии функции. В параметре шаблона может использовать как ключевого слово имя класса, так и *typename.* Обобщенная функция также называется шаблонной. Когда компилятор создает конкретную версию шаблонной функции, то создается ее специализации, которая также называется порожденной функцией *(generated function)*. Поскольку С++ не распознает символ конца строки в качестве признака конца инструкции, то *template* часть порожденной функции не может не находиться в одной строке с именем этой функции. В template инструкции можно объединить несколько обобщенных …, используя несколько элементов, разделенных запятой. Вручную перегруженная версия обобщенной функции называется явной специализацией. Создавая шаблонную функцию, вы разрешаете компилятору генерировать столько различных версий этой функции, сколько необходимо для обработки различных способов вызова этой функции. При наличии перегрузки обобщенной функции выбор делается в пользу конкретной ее версии.

Лекция №12л

В альтернативном синтаксисе объявления явной специализации функции можно использовать template<>.

Явная специализация шаблона позволяет спроектировать версию обобщённой функции в расчёте на некоторую уникальную ситуацию, чтобы воспользоваться преимуществами быстродействия программы только для одного типа данных, но если вам нужно иметь различные типы функций для разных типов данных, то стоит использовать перегруженные функции, а не шаблонные. Помимо создания явным образом перегруженных версий обобщённой функции можно также перегружать спецификацию шаблона. Для этого необходимо создать ещё одну версию шаблона функции, которая будет отличатся от остальных шаблонов списком параметров.

Можно определить для шаблонной функции другие шаблоны, имеющие тоже самое имя и типы параметров, но различные наборы аргументов. В шаблонных

функциях можно смешивать параметры типа с обобщёнными параметрами. При перегрузке функции в теле каждой из них обычно задаются различные действия. Обобщённая функция должна выполнять одно и тоже действие для всех её версий, а отличие будет только в типе данных. Перегрузка функций всегда подразумевает родство версий перегруженных функций.

**Обобщенные классы**

Для создания обобщённого класса необходимо определить все используемые им (классом) алгоритмы при этом реальный тип обрабатываемых данных в классе будет задан как параметр в момент создания объекта этого класса. Обобщённые классы необходимо использовать в том случае если в классе существует логика, которая одинакова для разных типов данных. Шаблонный класс может иметь несколько параметров типов, заданных в template спецификации в виде списка, разделённого запятыми. Одно из наибольших достоинств обобщённых классов состоит в том, что они позволяют 1 раз написать код, отладить его, а затем применять его к данным любого типа, не переписывая код для каждого конкретного типа данных. В template спецификации для обобщённого класса можно также задавать аргументы, не являющиеся типами. На тип параметров, которые не представляют типы, налагаются ограничения. В этом случае разрешено использовать только целочисленные типы, указатели и ссылки. Аргументы, которые передаются параметру, не являющемуся типом, должны содержать только целочисленную константу, либо указатель, либо ссылку на глобальную функцию или объект. Таким образом эти “нетиповые” параметры следует рассматривать как константы поскольку их значения не могут быть изменены. Поскольку параметры “нетипы” обрабатываются как константы, их можно использовать для указания размерности статического массива. Шаблонный класс может по умолчанию определять аргумент, соответствующий обобщённому классу. Тип, используемый по умолчанию, можно предусмотреть для наиболее употребительного типа данных, позволяя при этом создавать классы, при необходимости, указывая нужный параметр типа. Подобно функциям можно создавать специализации класса, работающие с определённым типом.

Лекция №13

Тип, используемый по-умолчанию можно предусмотреть для чего-то там, позволяя при этом пользователю при необходимости указывать тип. Явно задаваемая классификация типа для класса расширяет диапазон применения поскольку она позволяет легко обрабатывать один или 2 специфических случая, оставляя все остальные варианты для обработки компилятора.