**Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования**

**"Уфимский государственный авиационный технический университет"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Программирование

**Отчет по лабораторной работе № 6**

**Тема: «Введение в классы C++»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМИ-148 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Фаизова А.Н. |  |  |  |
| Принял | Гайнетдинова А.А. |  |  |  |

**Уфа 2018**

**Цель:***ознакомиться с описанием и использованием структур, а также понять принципы создания классов.*

**Теоретический материал**

Объектно-ориентированное программирование (ООП) — подход к программированию, при котором основными концепциями являются понятия объектов и классов.

Класс — это определяемый разработчиком тип данных.

Тип данных характеризуется:

* Способом хранения и представления этих данных.
* Назначением этих данных (профилем их использования).
* Набором действий, которые над этими данными могут производится.

Например, тип int предназначен для хранения целых чисел и подразумевает ряд операция над ними (+,-,\*,/,% и пр.).

*Класс* — это тип, описывающий устройство объектов, их поведение и способ представления.

*Объект* — сущность обладающая определённым поведением и способом представления, т. е. относящееся к классу (говорят, что объект — это экземпляр класса, если он к нему относится).

Класс можно сравнить с чертежом, согласно которому создаются объекты. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области решаемой задачи.

Описание класса начинается со служебного слова class, вслед за которым указывается имя класса. Затем в блоке следует описание класса и после закрывающей скобки блока ставится точка с запятой.

Описание класса состоит из полей и методов.

*Поля* (или свойства, в рамках C++ это можно считать синонимом) описывают то, какие данные смогут хранить экземпляры класса (т.е. объекты). Конкретные значения сохраняются уже внутри объектов. Поля объявляются в теле класса.

К полям внутри класса можно обращаться непосредственно по именам полей.

*Методы класса* — это функции, которые смогут применяться к экземплярам класса. Грубо говоря, метод — это функция объявленная внутри класса и предназначенная для работы с его объектами.

Методы объявляются в теле класса. Описываться могут там же, но могут и за пределами класса (внутри класса в таком случае достаточно представить прототип метода, а за пределами класса определять метод поставив перед его именем — имя класса и оператор ::(оператор расширения области видимости, используемый для получения доступа к элементам пространства имен.)).

Методы и поля входящие в состав класса называются членами класса. При этом методы часто называют функциями-членами класса.

Пример:

class Complex {

double img;

double real;

};

В примере описан класс Complex с двумя полями img и real.

Абстракция данных

Абстракция данных — это выделение существенных характеристик объекта, которые отличают его от прочих объектов, четко определяя его границы.

Абстракция данных в ООП предусматривает выделение характеристик, существенных в рамках решаемой задачи и рассматриваемой предметной области.

Например, создавая программу для автоматизации работы склада, важно учитывать вес контейнера, размер контейнера, его положение на складе, но совсем не важен цвет контейнера.

Вес, размер и положение — это поля будущего объекта-контейнера.

Методы, которые к этому объекту смогут применяться, возможны такие: создать в программе новый объект-контейнер, переместить объект на указанную позицию, удалить объект со склада, пометив занятое им место как свободное.

Ключевые черты ООП

*Инкапсуляция* — это принцип, согласно которому любой класс должен рассматриваться как чёрный ящик — пользователь класса должен видеть и использовать только интерфейсную часть класса (т. е. список декларируемых свойств и методов класса) и не вникать в его внутреннюю реализацию. Поэтому данные принято инкапсулировать в классе таким образом, чтобы доступ к ним по чтению или записи осуществлялся не напрямую, а с помощью методов. Принцип инкапсуляции (теоретически) позволяет минимизировать число связей между классами и, соответственно, упростить независимую реализацию и модификацию классов.

*Наследование* — это порождение нового класса-потомка от уже существующего класса-родителя. Класс-родитель называют также супер-классом, а класс-потомок — подклассом. Наследование происходит с передачей всех или некоторых полей и методов от класса-родителя к классу-потомку. В процессе наследования возможно, при необходимости, добавлять новые свойства и методы. Набор классов, связанных отношением наследования, называют иерархией.

*Полиморфизм* — это явление, при котором функции (методу) с одним и тем же именем соответствует разный программный код (полиморфный код) в зависимости от того, в каком контексте он вызывается (объектами какого класса или с какими параметрами).

Уровни доступа к членам класса

По уровню доступа все члены класса делятся на открытые (public), закрытые (private) и защищённые (protected).

Перед объявлением членов внутри класса ставятся соответствующие ключевые слова. Если такое слово не поставлено, то считается, что член объявлен с уровнем private. В примере выше класса Complex, соответственно, оба поля имеют уровень доступа private.

Члены объявленные как private доступны только внутри класса.

Члены объявленные как protected доступны внутри класса и внутри всех его потомков.

Члены объявленные как public доступны как внутри, так вне класса (в том числе в потомках).

Интерфейс класса

Все методы класса, находящиеся в его открытой части (с уровнем доступа public) называют интерфейсом класса

Интерфейс предназначен для взаимодействия класса с остальной программой. Зная методы интерфейса и их назначение можно использовать класс, не вникая в его внутреннее устройство. Соответственно, интерфейс реализует принцип инкапсуляции.

Конструктор и деструктор

При создании объектов одной из наиболее широко используемых операций, которая будет выполняться в программах, является инициализация элементов данных объекта. Чтобы упростить процесс инициализации элементов данных класса, C++ использует специальную функцию, называемую конструктором, которая запускается для каждого создаваемого вами объекта. Также C++ обеспечивает функцию, называемую деструктором, которая запускается при уничтожении объекта.

Конструктор представляет собой метод класса, который облегчает вашим программам инициализацию полей при создании объекта класса.

Конструктор имеет такое же имя, как и сам класс.

Конструктор не имеет возвращаемого значения (по сути, результатом его работы является ссылка на созданный объект).

Каждый раз, когда ваша программа создает объект, C++ вызывает конструктор класса, если подходящий (с соответствующими параметрами) конструктор существует.

Конструкторы относят к интерфейсу класса, чтобы с их помощью можно было создавать объекты данного класса из внешней части программы.

Конструктор по умолчанию

Конструктор по умолчанию — это конструктор без параметров.

Если он не задан явно и при этом не создано других конструкторов с параметрами, то конструктор по умолчанию создастся автоматически. При этом все свойства нового объекта не будут никак проинициализрованы (получат «мусорные» значения из памяти).

Деструктор

После того, как вы по ходу программы перестаёте пользоваться некоторым объектом, для него автоматически вызывается специальный метод-деструктор, если он описан в классе.

Представлять конструктор можно как функцию, которая помогает вам строить (конструировать) объект. Подобно этому, деструктор представляет собой функцию, которая помогает корректно уничтожать объект. Деструктор обычно используется, если при уничтожении объекта нужно освободить динамическую память, занимаемую объектом.

Деструктор имеет такое же имя, как и класс, но с символом тильды (~) в качестве префикса.

Деструктор не имеет возвращаемого значения и не имеет аргументов. Деструктор в классе всегда один (его нельзя перегружать). По умолчанию деструктор не создаётся, если он требуется, то его надо описывать явно.

Когда объявляется класс-наследник, то сразу после его имени ставится двоеточие и указывается уровень (или способ) наследования (public, private, protected), а затем имя класса родителя. Уже потом открывается блок с описанием класс-наследника.

Private-наследование

Те члены, что в родителе были protected и public, становятся private в потомке.

Protected-наследование

Те члены, что в родители были protected и public становятся protected в потомке. Такой вариант используется редко.

Public-наследование

Те члены, что в родители были protected и public, остаются, соответственно, protected и public в наследнике, сохраняя свой уровень доступа.

Если уровень явно не указан, то происходит private-наследование.

Можно говорить о том, что public-наследование — это наследование интерфейса, а private-наследование — это наследование реализации (т.е. все элементы наследуются, но становятся закрытыми от доступа из вне).

Перегрузка операторов

В С++ можно создавать методы со специальными именами вида: operator \*, где \* — это один из существующих операторов языка (например, +, -, /, \*, = и т.д.).

При создании такого метода, говорят, что оператор \* был перегружен, т.е. теперь его можно использовать для взаимодействия с объектами создаваемого класса. Перегружать можно многие, но не любые операторы.

Вызывать этот метод можно обращаясь к нему по указанному символу (в данном случае, \*). При этом аргументом метода будет считаться объект стоящий справа от символа, а вызываться метод будет для объекта, стоящего слева от символа.

Так, например, удобно для объектов своего класса использовать символ + в качестве метода, складывающего объекты, вместо именованного метода (например, sum).

Перегрузка операторов, как и перегрузка функций в целом, реализует принцип полиморфизма.

Пространство имен — это декларативная область, в рамках которой определяются различные идентификаторы (имена типов, функций, переменных, и т. д.). Пространства имен используются для организации кода в виде логических групп и с целью избежания конфликтов имен, которые могут возникнуть, особенно в таких случаях, когда база кода включает несколько библиотек. Все идентификаторы в пределах пространства имен доступны друг другу без уточнения. Как можно будет заметить в начале программ для удобства опущен стандартный префикс std (стандартная библиотека шаблонов находится именно в данном пространстве имен). Это делается для удобства чтения кода, а также для того, чтобы не усложнять код программ.

static\_cast — унарная операция приведения типов данных в С++

Операция static\_cast доступна только в языке C++. static\_cast может быть использована для преобразования одного типа в другой, но она не должна быть использована для выполнения недопустимого преобразования, например, преобразование значения в указатель или наоборот. Рекомендуется пользоваться операцией static\_cast, нежели Cи-стилем приведения, потому что static\_cast ограничивает недопустимое приведение типов и, следовательно — безопаснее.

Операция static\_cast, грубо говоря, — это шаблон функции, в которой необходимо явно указать тип данных для преобразования, то есть задать параметр шаблона.

static\_cast<dataType>(value);

Итак, в треугольных скобочках указывается тип данных, к которому необходимо преобразовать значение value, которое стоит в круглых скобочках.

**Ответы на контрольные вопросы**

1. Что такое класс данных?

Класс – это набор данных плюс набор функций для работы с этими данными. Классы моделирует объекты, которые встречаются в программировании, математике и т.п. Класс или объект класса можно представлять как нечто целое, имеющее внутреннюю структуру.

1. Что входит в состав класса?

В состав класса данные класса, также как и данные структур, называются полями. Функции, входящие в состав класса, называются методами.Члены класса – это переменные, содержащиеся внутри каждого объекта класса, как бы входящие в "конструкцию" класса.

1. Чем классы отличаются от структур и объединений?

Структуры (struct) и объединения (union) представляют собой частные случаи классов. Класс в C++ отличается от структуры только способом создания и тем фактом, что все поля структур по умолчанию считаются общедоступными (public), а все поля класса по умолчанию считаются приватными (private). То есть в структурах по умолчанию доступ ко всем полям разрешен, а в классах – запрещен.

Доступ в объединениях также устанавливается public, кроме того, в них вообще нельзя использовать спецификаторы доступа. Объединение не может участвовать в иерархии классов. Элементами объединения не могут быть объекты, содержащие конструкторы и деструкторы. Объединение может иметь конструктор и другие методы, только не статические. В анонимном объединении методы описывать нельзя.

1. Классы и структуры – что из них поддерживает наследование?

И классы, и структуры поддерживают наследование. Однако, структуры отличаются от классов тем, что доступ к элементам, а также базовый класс при наследовании по умолчанию считаются public.

1. Из каких двух частей состоит описание класса в C++?

Описание класса в C++ состоит из двух частей: описание интерфейса и реализация методов класса. Традиционно описание интерфейса класса помещается в заголовочный файл (h-файл), имя такого файла имеет расширение ".h". Описание интерфейса содержит описание переменных-членов класса и описание прототипов методов. Впрочем, некоторые простые методы могут быть реализованы непосредственно в h-файле; такую реализацию называют "внутристроковой" или "инлайновой" (inline).

При использовании инлайновых методов и функций их текст непосредственно вставляется в то место, где он используется. Объем кода при этом увеличивается (один и тот же фрагмент повторяется многократно при каждом использовании), зато программа получается значительно более быстрой: во первых, не используется механизм вызова функций; во вторых, и самое главное, – компилятор C++ получает возможность оптимизации кода, которая не ограничивается рамками одного метода. Реализация объемных (не инлайновых) методов класса выносится из заголовочного файла в файл реализации, который называют C++ файлом.

1. Для чего необходим и когда вызывается конструктор класса?

Конструктор - функция, предназначенная для инициализации объектов класса. Вызывается при создании нового объекта. Т.к. поля нельзя инициализировать непосредственно в объявлении класса, то конструктором часто пользуются для этих целей. Более того, в этом случае мы получаем более гибкий способ инициализации переменных.

1. Для чего необходим и когда вызывается деструктор класса?

Определяемый пользователем класс имеет конструктор, который обеспечивает надлежащую инициализацию. Для многих типов также требуется обратное действие. Деструктор обеспечивает соответствующую очистку объектов указанного типа. Имя деструктора представляет собой имя класса с предшествующим ему знаком "тильда" ~.

1. При помощи чего в классах обеспечивается инкапсуляция (сокрытие) внутренней структуры данных?

Сокрытие данных достигается посредством модификаторов доступа, каждый из которых определяет свой уровень доступности членов типа.

1. Опишите назначение модификаторов видимости private, protected и public. Общие черты и различия.

Классы позволяют пользователю управлять видимостью того, что лежит в основе их реализации. То, что в классе определено как public (общий, общедоступный), является видимым, т.е. доступным извне класса, а то, что определено какprivate (частный, собственный), – скрытым, т.е. изолированным внутри класса. Сокрытие данных – это один из основных принципов абстракции данных.

Public, private и protected — это модификаторы доступа, а не видимости, как ошибочно думают некоторые. Private члены видны снаружи класса, но не доступны.

Теперь кратко, кому какой доступ они предоставляют.

• Public — доступ открыт всем, кто видит определение данного класса.

• Protected — доступ открыт классам, производным от данного. То есть, производные классы получают свободный доступ к таким свойствам или метода. Все другие классы такого доступа не имеют.

• Private — доступ открыт самому классу (т.е. функциям-членам данного класса) и друзьям (friend) данного класса - как функциям, так и классам. Однако производные классы не получают доступа к этим данным совсем. И все другие классы такого доступа не имеют.

• В C++ существует public-наследование, private-наследование и protected-наследование. В зависимости от того, какой тип используется, изменяется доступ к членам базового класса для клиентов производного. В таблице сведена информация об этом изменении:

|  |  |  |  |
| --- | --- | --- | --- |
| Модификатор наследования | Модификатор доступа | | |
| public | private | protected |
| public-наследование | public | private | protected |
| private-наследование | private | private | private |
| protected-наследование | protected | private | protected |

1. Что такое наследование?

Каждый класс представляет собой некую сущность со своими свойствами, изменение которых позволяет создавать все многообразие объектов данного класса. Это входит в понятие объектно-ориентированного программирования (ООП) – полиморфизм.

Рассмотрим теперь такое понятие ООП, как наследование, то есть порождение нового класса от уже существующего. При этом, все элементы класса-предка, размещенные в зонах public и protected будут доступны в классе-потомке. Элементы класса, размещенные в зоне private доступны только самому классу.

Наследование является эффективным методом расширения функциональности класса и/или его изменения. Вместо того, чтобы создавать код с незначительными изменениями или малыми добавлениями полностью заново, в ООП существующий код берется без изменений как основа для нового класса и уже в код нового класса вносятся необходимые изменения. Таким образом, объем текста программы значительно сокращается, поскольку каждый участок кода присутствует в нем лишь один раз.

Для того, чтобы создать (породить) новый класс от уже существующего, необходимо после имени класса-потомка указать после двоеточия имя класса предка. Перед именем класса предка может также стоять модификатор видимости.

1. Как и для чего используется служебное слово virtual?

Ключевое слово virtual служит для информирования компилятора о том, что данная функция может быть переопределена дочерним классом.

При стандартной записи в потомке нельзя создать метод с таким же названием, что и в предке. Однако при построении класса-предка можно указать, какие методы могут быть переопределены в потомках при помощи служебного слова virtual.

Служебное слово virtual (виртуальная) показывает, что функция может иметь разные версии в разных производных классах, а выбор нужной версии при ее вызове - это задача транслятора. Тип функции указывается в базовом классе и не может быть переопределен в производном классе. Определение виртуальной функции должно даваться для того класса, в котором она была впервые описана (если только она не является чисто виртуальной функцией)

Если в производном классе не происходит переопределение функции (виртуальной или нет), то класс будет использовать версию функции базового класса. Если класс является частью длинной цепочки порождений, то будет использоваться самая последняя версия функции.

Следует также отметить, что в каждом классе почти всегда деструктор является виртуальным, поскольку он может и часто должен переопределяться в классе-потомке.

===================================================

Виртуальные функции — специальный вид функций-членов класса. Виртуальная функция отличается об обычной функции тем, что для обычной функции связывание вызова функции с ее определением осуществляется на этапе компиляции. Для виртуальных функций это происходит во время выполнения программы.

Для объявления виртуальной функции используется ключевое слово virtual. Функция-член класса может быть объявлена как виртуальная, если класс, содержащий виртуальную функцию, базовый в иерархии порождения; реализация функции зависит от класса и будет различной в каждом порожденном классе.

Виртуальная функция — это функция, которая определяется в базовом классе, а любой порожденный класс может ее переопределить. Виртуальная функция вызывается только через указатель или ссылку на базовый класс.

Определение того, какой экземпляр виртуальной функции вызывается по выражению вызова функции, зависит от класса объекта, адресуемого указателем или ссылкой, и осуществляется во время выполнения программы. Этот механизм называется динамическим (поздним) связыванием или разрешением типов во время выполнения.

Указатель на базовый класс может указывать либо на объект базового класса, либо на объект порожденного класса. Выбор функции-члена зависит от того, на объект какого класса при выполнении программы указывает указатель, но не от типа указателя. При отсутствии члена порожденного класса по умолчанию используется виртуальная функция базового класса.

1. Что такое абстрактный класс?

Абстрактный класс — это класс, который нельзя реализовать в виде объекта, т. к. в нем есть виртуальные функции.

При разработке иерархии классов всегда в качестве основы берут нечто, обладающее самыми общими атрибутами и методами, но не реализующееся в природе или в рассматриваемой области. Вот такая сущность называется абстрактным классом.

Абстрактный класс определяет интерфейс для переопределения производными классами. Абстрактный класс используют, когда необходимо создать семейство классов (много разновидностей монстров в игре) при этом было бы лучше вынести общую реализацию и поведение в отдельный класс.

Для того чтобы класс был абстрактным у него должна быть хотя бы одна чисто виртуальная функция. Прототип чисто виртуальной функции заканчивается присваиванием к нулю. Абстрактный класс никогда не реализуется в виде объекта. Другими словами, создать объект виртуального класса нельзя!

1. Почему конструктор не может быть виртуальным, а деструктор почти всегда является виртуальным?

Конструктор не может быть виртуальным, так как механизм диспетчеризации виртуального метода зависит от конструктора, который первым совершил инициализацию объекта.Конструктор для правильного построения объекта должен знать его истинный тип. Более того, конструктор - не совсем обычная функция. Он может взаимодействовать с функциями управления памятью, что невозможно для обычных функций. От обычных функций-членов он отличается еще тем, что не вызывается для существующих объектов. Следовательно, нельзя получить указатель на конструктор.

Деструкторы являются противоположностями конструкторови используются для очистки объектов после их использования. Обычно очистка состоит в удалении всех полей указателей в объекте.

**Индивидуальное задание №1**

Создать класс **Жилье**, имеющий поля:

- площадь,

- количество комнат.

Определить

- пустой конструктор и конструктор с полным заполнением полей,

- деструктор (с очисткой памяти),

- простые методы доступа к полям (set- и get-методы),

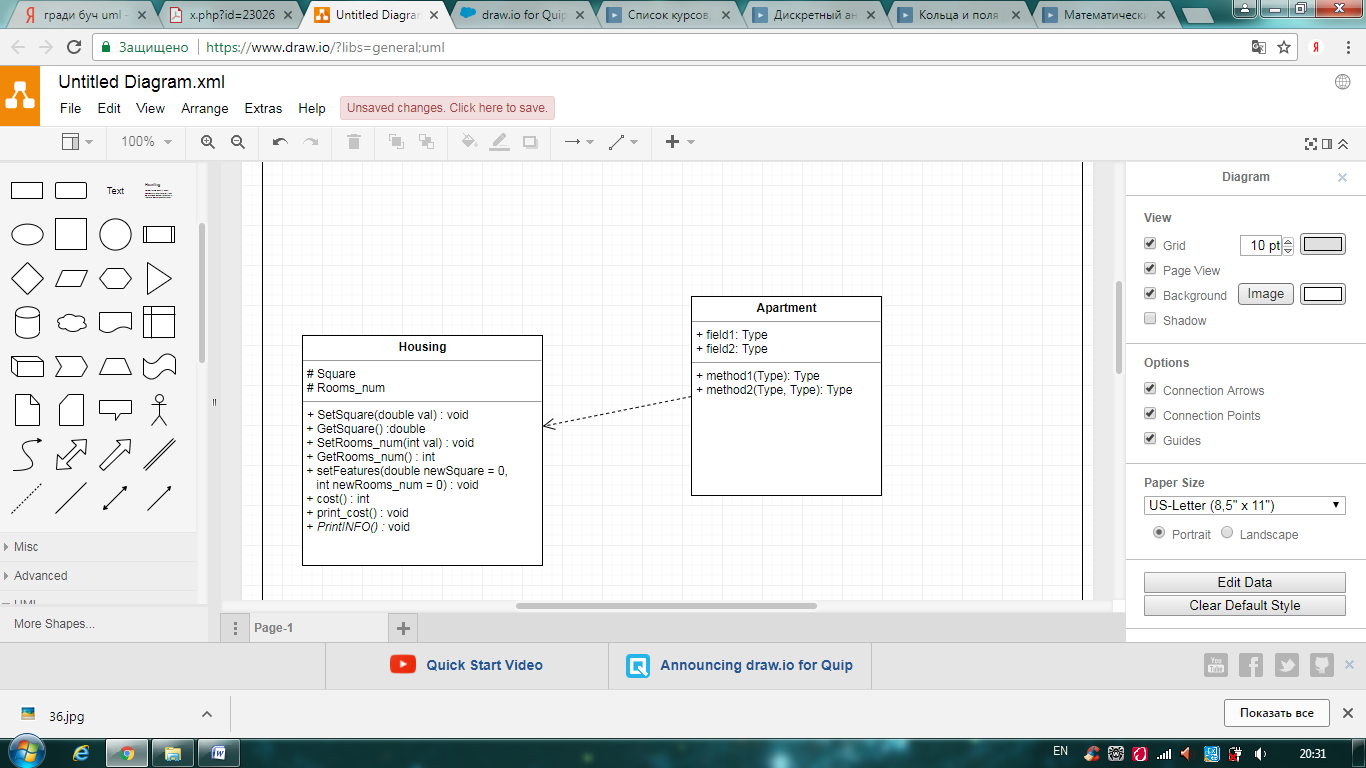
- метод вывода полной информации об объекте класса,

- метод расчета стоимости по цене за квадратный метр.

Класс должен быть разработан с соблюдением всех принципов объектно-ориентированного программирования

Создать объекты данного класса в функции main() и продемонстрировать работу всех методов класса.

Блок-схема



Исходный код программы

CHousing.h

#ifndefCHOUSING\_H

#define CHOUSING\_H

class CHousing

{

public:

// Пустойконструктор

CHousing();

// Kонструкторсполнымзаполнениемполей

CHousing(doubleSquare, intRooms\_num);

// Виртуальный деструктор

virtual ~CHousing();

// Возвращается значение поля f\_square

doubleGetSquare() { returnf\_Square; }

// Принимается значение val и записывается в поле f\_square

void SetSquare(double val) { f\_Square = val; }

// Возвращается значение поля f\_rooms\_num

int GetRooms\_num() { return f\_Rooms\_num; }

// Принимается значение val и записывается в поле f\_rooms\_num

void SetRooms\_num(int val) { f\_Rooms\_num = val; }

void setFeatures(double newSquare = 0, int newRooms\_num = 0);

// Mетод расчета стоимости по цене за квадратный метр

int cost();

void print\_cost();

// Mетод вывода полной информации об объекте класса

virtual void PrintINFO();

protected:

double f\_Square;

int f\_Rooms\_num;

private:

};

#endif // CHOUSING\_H

CHousing.cpp

#include <iostream>

#include "CHousing.h"

using namespace std;

CHousing::CHousing()

{

f\_Square = 0.0;

f\_Rooms\_num = 0;

}

CHousing::CHousing(double Square, int Rooms\_num)

{

f\_Square = Square;

f\_Rooms\_num = Rooms\_num;

}

CHousing::~CHousing()

{

//dtor

}

/// Set new features of the housing

/// @param newSquare- new value of housing square

/// @param newRooms\_num - new number of rooms

void CHousing::setFeatures(double newSquare, int newRooms\_num)

{

f\_Square = newSquare;

f\_Rooms\_num = newRooms\_num;

}

void CHousing::PrintINFO()

{

cout << endl << " -------------------------------------------------" << endl;

cout << " Square of housing: " << f\_Square << " (square meters);" << endl;

cout << " Number of rooms: " << f\_Rooms\_num << ";" << endl;

cout << " -------------------------------------------------" << endl;

}

int CHousing::cost()

{

return f\_Square \* 50000.0;

}

void CHousing::print\_cost()

{

cout << "Cost of housing " << CHousing::cost() << endl;

}

main.cpp

#include <iostream>

#include "CHousing.h"

using namespace std;

int main()

{

CHousing Dom1;

Dom1.setFeatures(123.4, 5);

Dom1.SetRooms\_num(6);

if (Dom1.GetSquare()<150.0){Dom1.SetSquare(100);}

Dom1.PrintINFO();

Dom1.print\_cost();

CHousing Dom2(99.2,5);

Dom2.PrintINFO();

Dom2.print\_cost();

double S;

int N;

int k=1;

CHousing Dom;

do {cout << "Square of housing: (square meters)" << endl;

cin >> S;

Dom.SetSquare(S);

cout << "Number of rooms: " << endl;

cin >> N;

Dom.SetRooms\_num(N);

Dom.PrintINFO();

Dom.print\_cost();

cout << " ##### Do you want to repeat? (Yes-1, No-0) #####" << endl;

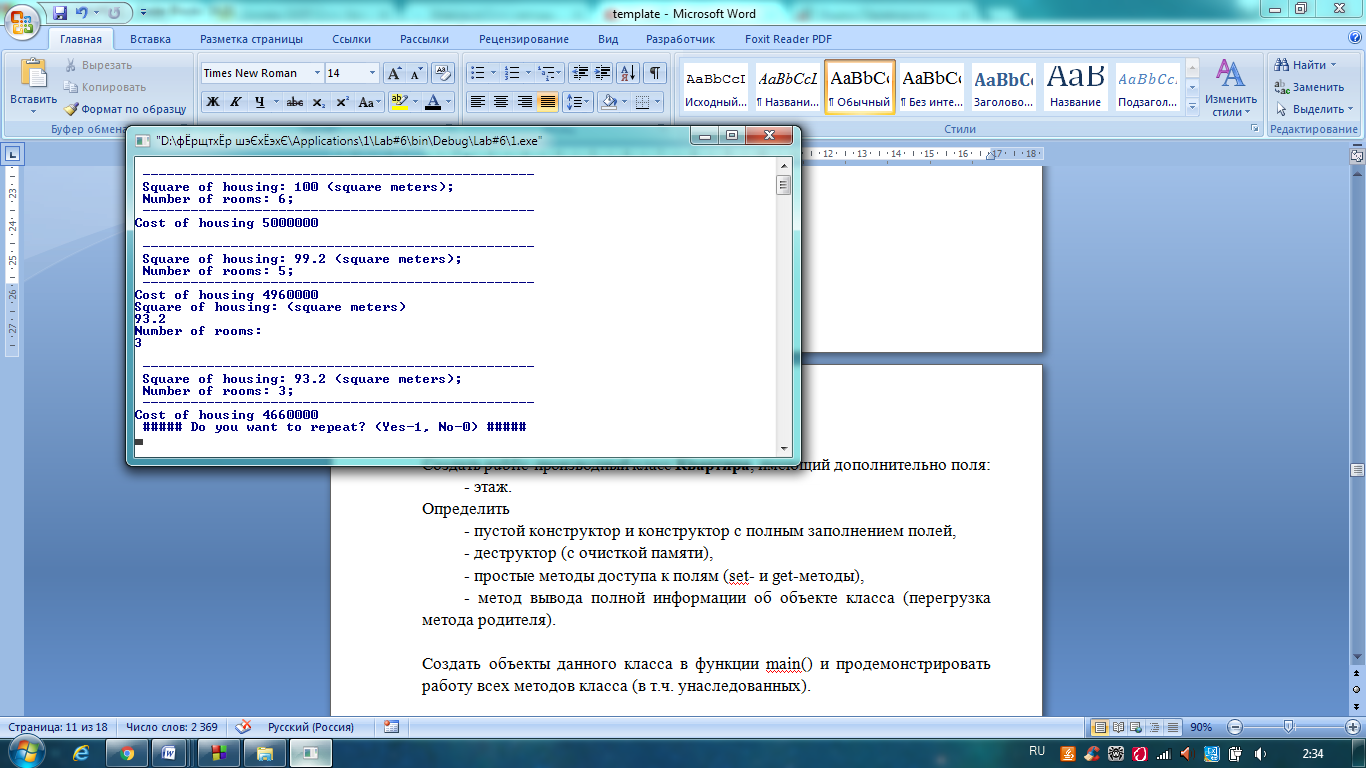
cin >> k;

}while (k==1);

return 0;

}

Пример выполнения программы



**Индивидуальное задание №2**

Создать public-производный класс **Квартира**, имеющий дополнительно поля:

- этаж.

Определить

- пустой конструктор и конструктор с полным заполнением полей,

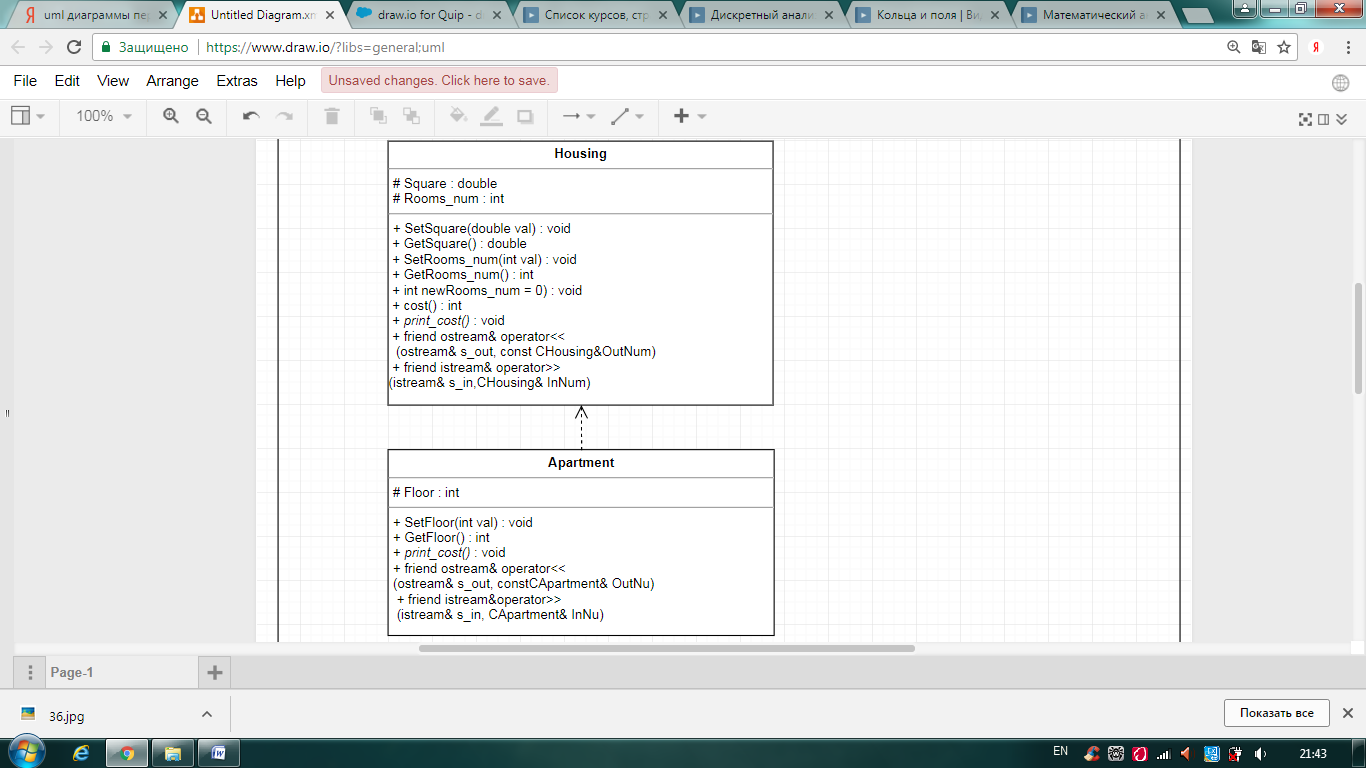
- деструктор (с очисткой памяти),

- простые методы доступа к полям (set- и get-методы),

- метод вывода полной информации об объекте класса (перегрузка метода родителя).

Создать объекты данного класса в функции main() и продемонстрировать работу всех методов класса (в т.ч. унаследованных).

Блок-схема



Исходный код программы

**CHousing.h**

#ifndefCHOUSING\_H

#defineCHOUSING\_H

/\*\* Директивы предпроцессора, запрещающие многократное вкл-е заголов.файла в проект. Если не определена константа, то определить её и добавить нижерасположенный код. Если константа уже определена, то переходим на #endif // CHOUSING\_H, и код ещё раз не вставляем

\*\*/

#include <iostream>

class CHousing

{

public:

// Пустойконструктор

CHousing();

// Kонструктор с полным заполнением полей

CHousing(double Square, int Rooms\_num);

// Виртуальный деструктор

virtual ~CHousing();

// Возвращается значение поля f\_Square

double GetSquare() { return f\_Square; }

// Принимается значение val и записывается в поле f\_Square

void SetSquare(double val) { f\_Square = val; }

// Возвращается значение поля f\_Rooms\_num

int GetRooms\_num() { return f\_Rooms\_num; }

// Принимается значение val и записывается в поле f\_Rooms\_num

void SetRooms\_num(int val) { f\_Rooms\_num = val; }

// Метод, изменяющийзначенияполей

void setFeatures(double newSquare = 0, int newRooms\_num = 0);

// Mетод расчета стоимости по цене за квадратный метр

int cost();

virtual void print\_cost();

// Mетод вывода полной информации об объекте класса

virtual voidPrintINFO();

// Перегрузка оператора потокового ввода/вывода

/\*\*Друж. фун-я для вывода получает ссылку на поток

и ссылку на переменную, которую нужно вывести\*\*/

friend std::ostream& operator<<(std::ostream& s\_out, const CHousing& OutNum);

/\*\* Возвращаем ссылку на поток ввода. Оператор ввода в поток

получает два параметра: ссылку на поток ввода

и ссылку на элемент, который нужно выводить \*\*/

friend std::istream& operator>>(std::istream& s\_in, CHousing& InNum);

protected:

double f\_Square;

int f\_Rooms\_num;

private:

};

#endif // CHOUSING\_H

**CHousing.cpp**

#include <iostream>

#include "CHousing.h"

using namespace std;

CHousing::CHousing()

{

f\_Square = 0.0;

f\_Rooms\_num = 0;

}

CHousing::CHousing(double Square, int Rooms\_num)

{

f\_Square = Square;

f\_Rooms\_num = Rooms\_num;

}

CHousing::~CHousing()

{

//dtor

}

/// Set new features of the housing

/// @param newSquare- new value of housing square

/// @param newRooms\_num - new number of rooms

void CHousing::setFeatures(double newSquare, int newRooms\_num)

{

f\_Square = newSquare;

f\_Rooms\_num = newRooms\_num;

}

//Метод вывода информации об объекте

voidCHousing::PrintINFO()

{

cout << endl << " -------------------------------------------------" << endl;

cout << " Square of housing: " << f\_Square << " (square meters);" << endl;

cout << " Number of rooms: " << f\_Rooms\_num << ";" << endl;

cout << " -------------------------------------------------" << endl;

}

int CHousing::cost()

{

returnf\_Square \* 50000.0;

}

// Метод расчета стоимости по цене за квадратный метр

voidCHousing::print\_cost()

{

// Конструкция cout возвращает ссылку на поток выводы

cout << "Cost of housing " << CHousing::cost() << endl;

}

ostream& operator<<(ostream& s\_out, const CHousing& OutNum)

{

s\_out << " Square of apartment: " << OutNum.f\_Square << "(square meters)\n" << " Number of rooms: " << OutNum.f\_Rooms\_num <<endl;

return s\_out;

}

istream& operator>>(istream& s\_in, CHousing& InNum)

{

cout << "Pleaseenter total area and number of rooms - ";

s\_in >> InNum.f\_Square >> InNum.f\_Rooms\_num;

return s\_in;}

**CApartment.h**

#ifndef CAPARTMENT\_H

#define CAPARTMENT\_H

#include <CHousing.h>

#include <iostream>

using namespace std;

class CApartment : public CHousing

{

public:

CApartment();

CApartment(double Square, int Rooms\_num, int Floor);

virtual ~CApartment();

// Возвращается значение поля f\_Floor

double GetFloor() { return f\_Floor; }

// Принимается значение val и записывается в поле f\_Floor

void SetFloor(double val) { f\_Floor = val; }

virtual void PrintINFO();

virtual void print\_cost();

friend std::ostream& operator<<(std::ostream& s\_out, const CApartment& OutNu);

friend std::istream& operator>>(std::istream& s\_in, CApartment& InNu);

protected:

int f\_Floor;

private:

};

#endif // CAPARTMENT\_H

**CApartment.cpp**

#include "CApartment.h"

CApartment::CApartment() : CHousing()

{

f\_Floor = 0;

}

CApartment::CApartment(double Square, int Rooms\_num, int Floor):CHousing(Square, Rooms\_num)

{

f\_Floor = Floor;

}

CApartment::~CApartment()

{

//dtor

}

void CApartment::print\_cost()

{

cout << "Cost of apartament " << CHousing::cost() << endl;

}

void CApartment::PrintINFO()

{

CHousing::PrintINFO();

Cout << “\n Floor of apartment” << f\_Floor;

}

ostream& operator<<(ostream& s\_out, const CApartment& OutNu)

{

s\_out << static\_cast <CHousing> (OutNu) << " Floor: " << OutNu.f\_Floor << endl << "===============================================" << endl;

return s\_out;

}

istream& operator>>(istream& s\_in, CApartment& InNu)

{

cout << "Please enter total area of apartment, number of rooms, floor - \n";

s\_in >> InNu.f\_Square >> InNu.f\_Rooms\_num >> InNu.f\_Floor;

return s\_in;

}

**main.cpp**

#include <iostream>

#include "CHousing.h"

#include "CApartment.h"

using namespace std;

int main()

{

CApartment Kvartira0;

cout<<Kvartira0<<endl;

CApartment Kvartira1(78.7, 3, 3);

Kvartira1.SetFloor(2);

Kvartira1.SetSquare(80.2);

cout<<Kvartira1<<endl;

CApartment Kvartira2;

cin>>Kvartira2;

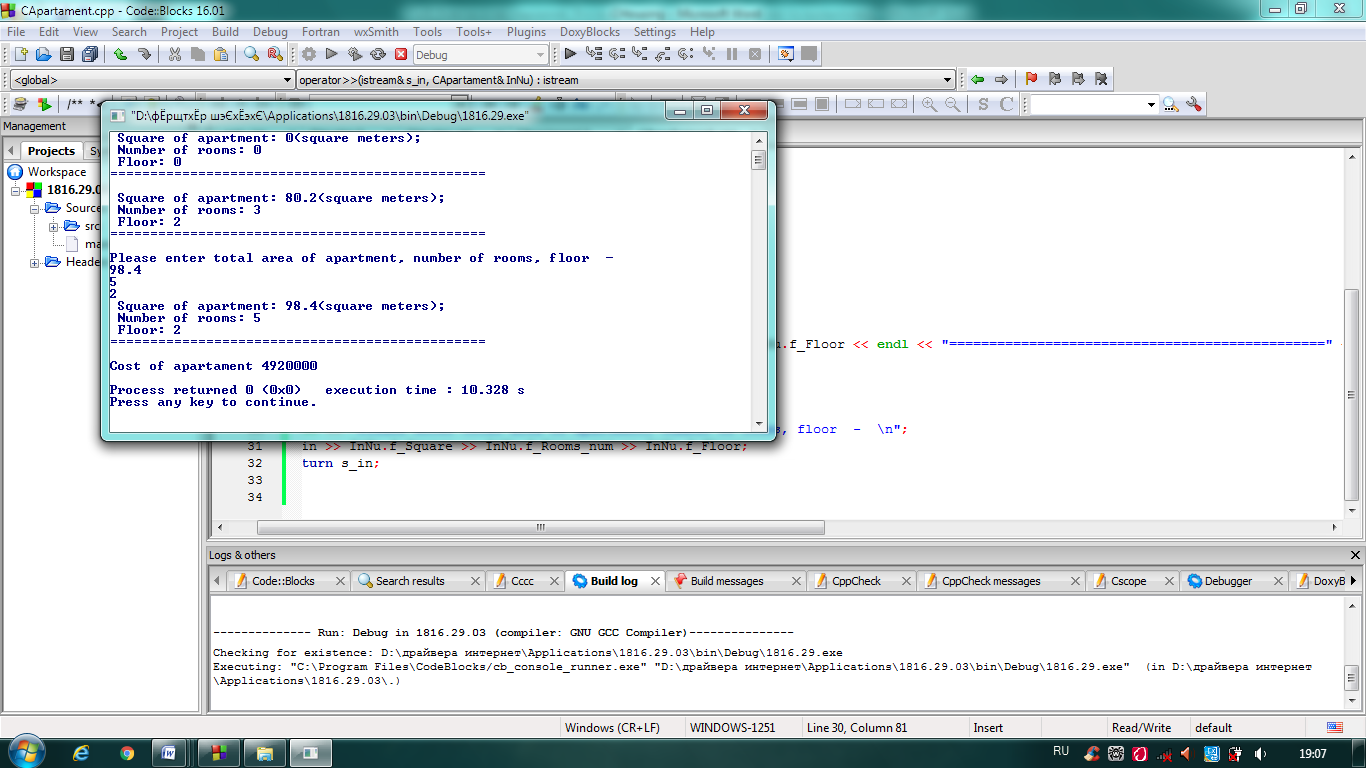
cout<<Kvartira2<<endl;

Kvartira2.print\_cost();

return 0;

}

Пример выполнения программы



**Вывод**

*На данной лабораторной работе мы ознакомились с описанием и использованием структур, а также изучили принципы создания классов.*

**Список использованной литературы**

1. Введение в классы С++[Электронный ресурс] / БикмеевА. Т. // <http://bikmeyev-at.ugatu.su/students/CPP/LR201/index15.html> (Дата обращения 29.03.2018).
2. Бьерн Страуструп. Язык программирования C++. Специальное издание. Пер. с англ. — М.: Издательство Бином, 2011 г. — 1136 с: ил.
3. Ввод данных в поток у наследников класса //

<http://www.cyberforum.ru/cpp-beginners/thread1260762.html> (Дата обращения 29.03.2018).