**Лекция. КЛАССЫ.**

Класс - это спецификация типа данных, определенного вами. Он может содер­жать элементы данных, которые могут быть как переменными базовых типов С++, так и других определенных пользователем типов. Элементы данных класса могут быть отдельными элементами данных, массивами, указателями, массивами указателей почти любого рода, объектами других классов, так что в вашем распоряжении практически неограниченная гибкость в отношении того, что можно включать в ваши типы данных. Класс также может содержать функции, которые оперируют объектами класса, обращаясь к элементам данных, которые они включают в себя. Таким образом, класс комбинирует определение элементарных данных, из которых состоит объект, и средства манипулирования данными, относящимися к индивидуальным объектам класса.

Так же, как объявление структуры, так же и объявление класса не приводит к выделению какой-либо памяти. Для использования класса нужно объявить переменную этого типа класса.

В C++ переменная класса называется **экземпляром**или**объектом класса**. Точно так же, как определение переменной фундаментального типа данных (например, int x) приводит к выделению памяти для этой переменной, так же и создание объекта класса) приводит к выделению памяти для этого объекта.

Помимо хранения данных, которые называют **полями**, классы также могут содержать и функции. Функции, определённые внутри класса, называются **функциями-членами** или **методами**. Методы могут быть определены внутри или вне класса.

Точно так же, как к членам структуры, так и к членам (переменным и функциям) класса доступ осуществляется через **оператор выбора членов** (**.**).

Определение класса

Рассмотрим пример класса— класс ящиков. Вы определили тип данных *СВох*, используя ключевое слово *class*, следующим образом:

*class СВох*

*{*

*public:*

*double m\_Length; //длина ящика*

*double m\_Width; //ширина ящика*

*double m\_Height; //высота ящика*

*};*

Имя класса следует за ключевым словом *class*, и между фигурными скобками определены три поля. Поля определены в классе с использованием операторов объявления, а все определение класса завершается точкой с запятой. Имена всех членов класса локальны по отношению к нему. Поэтому вы можете использовать те же имена еще где-то в программе, и это не вызовет никаких проблем.

Управление доступом в классе

**Спецификатор доступа** определяет, кто имеет доступ к членам этого спецификатора. Каждый из членов «приобретает» уровень доступа в соответствие со спецификатором доступа (или, если он не указан, в соответствие со спецификатором доступа по умолчанию).

**В C++ есть 3 уровня доступа**:

**public**: делает члены открытыми;

**private**: делает члены закрытыми;

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

**Открытые члены**(или ещё **«члены public»**) — это члены структуры или [**класса**](https://ravesli.com/urok-113-klassy-obekty-i-metody-klassov/), к которым можно получить доступ извне этой же структуры или класса.

**Закрытые члены**(или ещё **«члены private»**) — это члены класса, доступ к которым имеют только другие члены этого же класса.

Члены класса являются закрытыми по умолчанию, мы можем сделать их открытыми, используя **ключевое слово** **public**.

Классы могут использовать (и активно используют) сразу несколько спецификаторов доступа для установки уровней доступа для каждого из своих членов. Обычно переменные-члены являются закрытыми, а методы — открытыми.

Открытые члены классов составляют **открытый** (или ещё **«public»**) **интерфейс**. Поскольку доступ к открытым членам класса может осуществляться извне класса, то открытый интерфейс и определяет, как программы, использующие класс, будут взаимодействовать с этим же классом.

**Класс по умолчанию устанавливает всем своим членам спецификатор доступа private. Структура же по умолчанию устанавливает всем своим членам спецификатор доступа public**.

## Инкапсуляция

В объектно-ориентированном программировании **инкапсуляция** (или ещё **«сокрытие информации»**) — это процесс скрытого хранения деталей реализации объекта. Пользователи обращаются к объекту через открытый интерфейс.

В C++ инкапсуляция реализована через [**спецификаторы доступа**](https://ravesli.com/urok-114-spetsifikatory-dostupa-public-i-private/). Как правило, все переменные-члены класса являются закрытыми (скрывая детали реализации), а большинство методов являются открытыми (с открытым интерфейсом для пользователя). Хотя требование к пользователям использовать публичный интерфейс может показаться более обременительным, нежели просто открыть доступ к переменным-членам, но, на самом деле, это предоставляет большое количество полезных преимуществ, которые улучшают возможность повторного использования кода и его поддержку.

Инкапсуляция имеет много преимуществ, основное из которых — это использовать класс без необходимости понимания его реализации. Инкапсулированные классы:

   проще в использовании;

   уменьшают сложность программы;

   защищают и предотвращают данные от неправильного использования;

   легче изменять;

   легче отлаживать.

Это значительно облегчает использование классов.

Определение объектов класса

Вы объявляете объекты класса точно таким же образом, как и объекты базовых типов, поэтому вы можете объявлять объекты типа класса СВох с помощью следующих операторов:

*СВох boxl; // Объявление boxl типа СВох*

*СВох bох2; // Объявление bох2 типа СВох*

Оба объекта - *box1* и *bох2* - конечно же, будут иметь свои собственные поля.

**Доступ к полям класса**

Можно обращаться к полям класса. используя операцию прямого выбора члена.

*boxl .m\_Height = 18.0; //Установка значения поля данных*

Можно таким образом обратиться к полям в функции, находящейся вне класса, потому что *m\_Height* определен со спецификатором доступа *public*.

Первое применение классов.

//listing 36

//Операция прямого выбора поля

//Создание и использование ящиков

#include <iostream.h>

class CBox //Определение класса в глобальном контексте

{

public:

double m\_Lenght; //Длина ящика

double m\_Widht; //Ширина ящика

double m\_Height; //Высота ящика

};

int main ()

{

CBox box1; //объявление box1 типа CBox

CBox box2; //объявление box2 типа CBox

double boxVolume = 0.0; //сохранить объем ящика

box1.m\_Height = 18.0; //определить значения полей объекта box1

box1.m\_Lenght = 78.0;

box1.m\_Widht = 24.0;

box2.m\_Height = box1.m\_Height - 10;

//определить поля объекта box2 в терминах box1

box2.m\_Lenght = box1.m\_Lenght/2.0;

box2.m\_Widht = 0.25\*box1.m\_Lenght;

//Вычислить объем box1

boxVolume = box1.m\_Height\*box1.m\_Lenght\*box1.m\_Widht;

cout <<endl <<"V box1 =" <<boxVolume;

//Вычислить сумму сторон box2

cout <<endl <<"box2 sum =" <<box2.m\_Height + box2.m\_Lenght + box2.m\_Widht;

//Отобразить размер ящика в памяти

cout <<endl <<"razm CBox " <<sizeof box1 <<" b ";

cout <<endl;

return 0;

}

Результат

V box1 =33696

box2 sum =66.5

razm CBox 24 b

При вводе кода функции *main ()* вы должны наблюдать подсказки редактора со списком имен членов при каждом вводе операции выбора члена вслед за именем объекта класса. Затем вы можете выбирать нужный член в списке двойным щелчком мыши. Наведение курсора мыши на любую переменную в вашем коде приведет к отображению ее типа.

Описание полученных результатов

Поскольку все примеры пока оформлены в виде консольных приложений, про­ект должен быть определен соответствующим образом. Все здесь работает так, как и можно ожидать, основываясь на опыте применения структур. Определение класса находится вне функции *main ()*, и потому относится к глобальному контексту. Это позволяет вам объявлять объекты в любой функции программы и отображает класс на вкладке Class View (Представление классов), как только программа будет скомпилирована.

Вы объявили два объекта типа *СВох* внутри функции main () — boxl и box2. Конечно, как это было бы и с переменными базовых типов, объекты boxl и box2 локальны по отношению к main (). Объекты типов классов подчиняются тем же правилам видимости объявленных переменных, что и переменные базовых типов.

Первые три оператора присваивания устанавливают значения полей boxl. Вы определяете значения полей box2 в терминах полей boxl в следующих трех операторах присваивания.

Затем идет оператор, который вычисляет объем boxl как произведение его трех полей, после чего полученное значение выводится на экран. Далее выводится сумма полей box2, для чего применяется выражение, суммирующее поля непосредственно в операторе вывода. Финальное действие программы - вывод количества байт памяти, занятых boxl, которое возвращает операция sizeof.

Последняя строка показывает, что объект boxl занимает 24 байта памяти, что объ­ясняется наличием 3 полей по 8 байт каждый.

Методы класса.

Метод класса – это функция, определение или прототип которой находятся внутри определения класса. Она оперирует любым объектом класса, членом которого является, и имеет доступ ко всем членам класса этого объекта.

Добавление метода к классу СВох.

Чтобы увидеть, как можно обращаться к полям класса изнутри метода, рассмотрим пример, расширяющий класс СВох за счет включения в него метода, который вычисляет объем объекта СВох.

//listing 37

//Вычисление объема ящика с помощью функции-метода

#include <iostream.h>

class CBox //Определение класса в глобальном контексте

{

public:

double m\_Lenght; //Длина ящика

double m\_Widht; //Ширина ящика

double m\_Height; //Высота ящика

**//Функция для вычисления объема ящика**

**double Volume()**

**{**

**return m\_Lenght\*m\_Widht\*m\_Height;**

**}**

};

int main ()

{

CBox box1; //объявление box1 типа CBox

CBox box2; //объявление box2 типа CBox

double boxVolume = 0.0; //сохранить объем ящика

box1.m\_Height = 18.0; //определить значения полей объекта box1

box1.m\_Lenght = 78.0;

box1.m\_Widht = 24.0;

box2.m\_Height = box1.m\_Height - 10;

//определить поля объекта box2 в терминах box1

box2.m\_Lenght = box1.m\_Lenght/2.0;

box2.m\_Widht = 0.25\*box1.m\_Lenght;

//Вычислить объем box1

**boxVolume = box1.Volume();**

**cout <<endl <<"V box1 =" <<boxVolume;**

**//Вычислить объем box2**

**cout <<endl <<" V box2 =" <<box2.Volume();**

//Отобразить размер ящика в памяти

cout <<endl <<"razm CBox " <<sizeof box1 <<" b ";

cout <<endl;

return 0;

}

V box1 =33696

V box2 =6084

razm CBox 24 b

Описание полученных результатов

Новый код, добавленный к определению класса *СВох*, выделен полужирным шрифтом. В нем - только определение функции *Volume ()*, которая является методом класса. Она также имеет некоторый атрибут доступа, подобно данным-членам, а именно - *public*. Это потому, что каждый член класса, следующий за атрибутом доступа, имеет этот атрибут до тех пор, пока в определении класса не встретится другой. Функция *Volume ()* возвращает объем объекта *СВох* как значение типа *double*. Выражение в операторе *return -* это просто произведение трех полей класса.

Нет необходимости определять поля класса как параметры при обращении к ним из методов. При выполнении функции неопределенные имена членов автоматически ссылаются на поля текущего объекта.

Метод *Volume ()* используется в выделенных операторах *main ()* после ини­циализации полей (как в первом примере). Использование одного и того же имени переменной в *main ()* не вызывает конфликтов или каких-то других проблем. Вы можете вызывать методы определенного объекта, указывая имя объекта, точку, а за ней - имя метода. Как уже отмечалось, функция автоматически обращается к полям объекта, для которого она вызвана, поэтому первое применение *Volume ()* вычисляет объем *boxl*. Использование только имени поля всегда будет ссылаться на объект, для которого метод был вызван.

Метод вызывается второй раз непосредственно в операторе вывода, чтобы вычислить объем *box2*.

Обратите внимание, что объект *СВох* по-прежнему занимает в памяти столько же байт, сколько и ранее. Добавление метода к классу не повлияло на размер объектов. Очевидно, что метод должен находиться где-то в памяти, но существует лишь одна его копия, независимо от того, сколько объектов вы создаете, и память, занятая функцией, не считается при выполнении операции sizeof, определяющей количество байт памяти, занятых объектом. Имена полей класса в методе автоматически ссылаются на поля конкретного объекта, используемого для вызова функции, и функция может быть вызвана только с определенным объектом класса. В данном случае это делается через операцию прямого обращения к члену, следующего за именем объекта.

Если вы попробуете вызвать метод, не указывая имени объекта, программа компи­лироваться не будет.

Расположение определения метода

Определение метода должно быть помещено внутрь определения класса. Если вы хотите разместить его вне определения класса, то внутри класса потребуется указать его прототип. Если переписать предыдущий класс с определением функции вне класса, он будет выглядеть следующим образом:

*class СВох // Определение класса в глобальном контексте*

*{*

*public:*

*double m\_Length; // Длина ящика*

*double m\_Width; // Ширина ящика*

*double m\_Height; // Высота ящика*

*double Volume(void);*

*}*

Но теперь вы должны написать определение функции, однако поскольку оно появляется вне определения класса, должен существовать какой-то способ сообщить компилятору, что эта функция относится к классу СВох. Это делается за счет

снабжения имени функции префиксом - именем класса и разделением их операцией разрешения контекста, обозначаемой двумя двоеточиями (::). Определение функции теперь будет выглядеть так:

*// Функция для вычисления объема ящика*

*double СВох::Volume()*

*{*

*return m\_Length\*m\_Width\*m\_Height;*

*}*

Она обеспечит тот же вывод, что и в предыдущем примере, однако это не будет в точности той же программой. Во втором случае все вызовы функции трактуются способом, который вам уже известен. Однако когда вы определяете функцию внутри определения класса, то компилятор неявно трактует ее как встроенную(inline) функцию.

Встроенные функции

Со встроенными функциями компилятор пытается расширить код телом функции в месте ее вызова. Это позволяет избежать накладных расходов, связанных с вызовом функции, и потому повышает скорость выполнения вашего кода.

Разумеется, компилятор гарантирует, что расширение вызова телом встроенной функции не вызовет никаких проблем с именами переменных или областью видимости.

Компилятор не всегда может вставить код встроенной функции в место ее вызова (например, в случае рекурсивных функций или функций, для которых вы получаете адрес), но в основном это работает. Лучше всего использовать это для очень коротких, простых функций вроде нашей функции *Volume ()* в классе *СВох*, потому что такая функция выполняется быстрее, а встраивание ее кода не увеличивает существенно размера исполняемого модуля.

При определении функции вне определения класса компилятор трактует функцию как нормальную, и ее вызов работает обычным образом; однако, также можно сообщить компилятору, что если возможно, то вы бы предпочли, чтобы функция трактовалась как встроенная. Это делается простым добавлением

ключевого слова *inline* в начале заголовка функции. Таким образом, для этой функции определение должно выглядеть следующим образом:

*// Функция для вычисления объема ящика*

*inline double СВох::Volume()*

*{*

*return m\_Length\*m\_Width\*m\_Height;*

*}*

При таком определении функции программа будет точно такой же, как и рань­ше. Но так можно поместить определение функции-члена за пределами определения класса, если вы этого хотите, и, тем не менее, воспользоваться преимуществами в скорости выполнения, которые дает встраивание.

Вы можете применить ключевое слово inline к обычным функциям вашей программы, которые не имеют отношения к классам, и получить тот же эффект. Помните, однако, что лучше всего это работает с короткими простыми функциями.

Конструкторы классов

В предыдущем примере программы вы объявили объекты *СВох*, *boxl* и *box2*, а затем старательно прошли по всем полям каждого объекта, чтобы присвоить им начальные значения. Это неудовлетворительно с нескольких точек зрения. Прежде всего, очень легко пропустить инициализацию каких-то полей, в частности в классах, у которых их намного больше, чем в нашем классе *СВох*.

Инициализация полей некоторых объектов сложных классов может потребовать многих страниц кода, состоящих из операторов присваивания. Конечное ограничение такого подхода проявляется, когда определяются поля класса, которые не имеют атрибута *public* - у вас нет доступа к ним извне класса. Должен существовать другой, лучший способ, и конечно же, он есть - это конструктор.

Что такое конструктор?

Конструктор класса - это специальная функция класса, которая вызывается при создании нового объекта этого класса. Таким образом, она предоставляет возможность инициализировать объекты во время их создания и гарантировать, что все поля будут иметь корректные значения. Класс может иметь несколько конструкторов, позволяя создавать объекты различными способами.

Конструкторы всегда называются по имени класса, в котором определены. Функция *СВох ()*, например, конструктор нашего класса *СВох* . Конструктор не имеет типа возврата. основное назначение конструктора класса – присваивать начальные значения элементам данных класса, и никакой тип возврата не требуется и не допускается.

Добавление конструктора к классу CBox

//listing 38

//Использование конструктора

#include <iostream.h>

class CBox //Определение класса в глобальном контексте

{

public:

double m\_Lenght; //Длина ящика

double m\_Widht; //Ширина ящика

double m\_Height; //Высота ящика

**//определение конструктора**

**CBox (double lv, double bv, double hv)**

**{**

**cout <<endl << "constructor work";**

**m\_Lenght = lv; //установить значения полей**

**m\_Widht = bv;**

**m\_Height = hv;**

**}**

//Функция для вычисления объема ящика

double Volume()

{

return m\_Lenght\*m\_Widht\*m\_Height;

}

};

int main ()

{

//объявление и инициализация box1 типа CBox

**CBox box1 (78.0,24.0,18.0);**

//объявление и инициализация cigarBox типа CBox

**CBox cigarBox (8.0,5.0,1.0);**

double boxVolume = 0.0; //сохранить объем ящика

boxVolume = box1.Volume(); //Вычислить объем box1

cout <<endl

<<"V box1 =" <<boxVolume;

cout <<endl

<<" V cigarBox ="

<<cigarBox.Volume(); //Вычислить объем cigarBox

cout <<endl;

return 0;

}

Результат

constructor work

constructor work

V box1 =33696

V cigarBox =40

Описание полученных результатов

Конструктор *СВох ()* принимает три параметра типа *double*, соответствующие начальным значениям членов *m\_Length*, *m\_Width* и *m\_Height* объекта *СВох*. Первый оператор конструктора выводит сообщение, чтобы было видно, когда он был вызван.

Внутри *main ()* вы объявляете объект *box1* с инициализирующими значениями последовательно для полей *m\_\_Length, m\_Width* и *m\_Height*. Они указаны в скобках, следующих за именем объекта. Здесь используется нотация функций для инициализации, которая, также может быть применена для инициализации обычных значений базовых типов. Вы также объявляете второй объект *СВох* по имени *cigarBox*, который имеет инициализирующие значения. Объем *box1* вычисляется с применением метода Volume (), как и в предыдущем примере, после чего отображается на экране. Точно так же отображается значение объема cigarBox. Вывод этого примера будет таким:

*constructor work*

*constructor work*

*V box1 =33696*

*V cigarBox =40*

Первые две строки выводят вызовы конструктора *СВох ()* - по одному для каждого объявленного объекта. Конструктор, добавленный в определение класса, вызывается автоматически при объявлении объекта СВох, поэтому оба объекта инициализируются значениями, указанными в объявлении. Они передаются конструктору в виде аргументов, в той последовательности, как записаны в объявлении.

Конструктор по умолчанию

Конструктор, который не имеет параметров (или имеет параметры, все из которых имеют [значения по умолчанию](https://ravesli.com/urok-103-parametry-argumenty-po-umolchaniyu/)), называется конструктором по умолчанию. Он вызывается, если пользователем не указаны значения для инициализации.

Попробуем модифицировать последний пример, добавив объявление *box2*, которое было в предыдущем примере:

*СВох box2; // Объявление box2 типа СВох*

Здесь мы оставляем *box2* без инициализирующих значений. Когда вы попытаетесь пересобрать эту версию программы, то получите сообщение об ошибке:

*error С2512: 'СВох': no appropriate default constructor available*

**ошибка** C2512: 'CBox': нет **доступных** конструкторовпо **умолчанию**

Это значит, что компилятор ищет конструктор по умолчанию для *box2* (также называемый конструктором без аргументов, поскольку при вызове никаких аргумен­тов не требует), поскольку вы не указали никаких инициализирующих значений для полей.

Конструктор по умолчанию - это конструктор, не требующий аргументов, и он может либо не иметь параметров в своей спецификации, либо быть таким. у которого аргументы необязательны. Но этот оператор отлично удовлетворял компилятор в предыдущих задачах, почему же он не работает здесь?

Ответ состоит в том, что в предыдущем примере использовался конструктор по умолчанию без аргументов, предоставленный компилятором, так как не был определен свой конструктор. Поскольку в последнем примере конструктор определен, компилятор предполагает, что вы достаточно компетентны и обо всем позаботитесь сами, и не создает своего конструктора по умолчанию. Поэтому, если необходимо использовать объявления объектов *СВох* без инициализирующих значений, то нужно предусмотреть конструктор по умолчанию самостоятельно.

В простейшем случае конструктор по умолчанию это просто конструктор, не принимающий никаких аргументов.

*CBox () //Конструктор по умолчанию*

*{} //Полностью свободен от кода*

**Использование конструктора по умолчанию.**

Добавим версию конструктора по умолчанию к последнему примеру вместе с объявлением box2, а также с исходными присваиваниями полям box2.

Расширим конструктор по умолчанию, чтобы было видно, когда он вызван.

//listing 39

//Добавление и использование конструктора по умолчанию

#include <iostream.h>

class CBox //Определение класса в глобальном контексте

{

public:

double m\_Lenght; //Длина ящика

double m\_Widht; //Ширина ящика

double m\_Height; //Высота ящика

//определение конструктора

CBox (double lv, double bv, double hv)

{

cout <<endl << "constructor work";

m\_Lenght = lv; //установить значения полей

m\_Widht = bv;

m\_Height = hv;

}

**//определение конструктора по умолчанию**

**CBox ()**

**{**

**cout <<endl << "constructor by default work";**

**}**

//Функция для вычисления объема ящика

double Volume()

{

return m\_Lenght\*m\_Widht\*m\_Height;

}

};

int main ()

{

CBox box1 (78.0,24.0,18.0); //объявление и инициализация CBox

CBox box2; //объявление box2 без инициализ

CBox cigarBox (8.0,5.0,1.0); //объявление и инициализация double boxVolume = 0.0; //сохранить объем ящика

boxVolume = box1.Volume(); //Вычислить объем box1

cout <<endl

<<"V box1 =" <<boxVolume;

//определение полей box2

box2.m\_Height = box1.m\_Height – 10;

box2.m\_Lenght = box1.m\_Lenght / 2.0;

box2. m\_Widht = 0.25\* box1.m\_Lenght;

cout <<endl

<<" V box2 ="

<< box2.Volume(); //Вычислить объем box2

cout <<endl

<<" V cigarBox ="

<<cigarBox.Volume(); //Вычислить объем cigarBox

cout <<endl;

return 0;

}

Результат

constructor work

constructor by default work

constructor work

V box1 =33696

V box2 = 6084

V cigarBox =40

Описание полученных результатов

Теперь, когда вы включили собственную версию конструктора по умолчанию, нет никаких сообщений об ошибках от компилятора, и все работает. Программа выдает следующий результат:

*constructor work*

*constructor by default work*

*constructor work*

*V box1 =33696*

*V box2 = 6084*

*V cigarBox =40*

Все, что делает конструктор по умолчанию - это отображает сообщение. Очевидно, что он вызывается в точке объявления объекта *box 2*. Вы также получаете корректное значение объема всех трех объектов *СВох*, так что остальная часть программы работает должным образом.

Последний пример включает два конструктора, отличающихся списками параметров. Один имеет три параметра типа *double*, а другой вообще не имеет параметров.

Присваивание параметрам в классе значения по умолчанию

Можно определить значения параметров по умолчанию в прототипе метода класса, включая конструкторы. Если вы поместите определение метода внутрь определения класса, то можете указать значения ее параметров по умолчанию прямо в заголовке функции. Если же в определение класса включен только прототип функции, то в этом прототипе должны присутствовать значения параметров по умолчанию.

Использование значений по умолчанию для аргументов конструктора.

Пусть значение размера объекта *СВох* по умолчанию должно быть равно одиночному ящику кубической формы со всеми сторонами длиной 1.

//listing 40

//Использование значений по умолчанию для аргументов конструктора

#include <iostream.h>

class CBox //Определение класса в глобальном контексте

{

public:

double m\_Lenght; //Длина ящика

double m\_Widht; //Ширина ящика

double m\_Height; //Высота ящика

**//определение конструктора**

**CBox (double lv =1.0, double bv =1.0, double hv = 1.0)**

**{**

**cout <<endl << "constructor work";**

**m\_Lenght = lv; //установить значения полей**

**m\_Widht = bv;**

**m\_Height = hv;**

**}**

//Функция для вычисления объема ящика

double Volume()

{

return m\_Lenght\*m\_Widht\*m\_Height;

}

};

int main ()

{

CBox box2; //объявление box2 без инициализ

cout <<endl

<<" V box2 ="

<< box2.Volume(); //Вычислить объем box2

cout <<endl;

return 0;

}

Результат

constructor work

V box2 = 1

Описание полученных результатов

Вы объявили одну неинициализированную переменную *СВох* по имени *box2*, потому что для целей демонстрации этого достаточно. Эта версия программы выдает следующий вывод:

*constructor work*

*V box2 = 1*

Это доказывает, что конструктор со значениями параметров по умолчанию выполняет свою работу, устанавливая значения объектов, для которых не заданы начальные значения.

Использование списка инициализации в конструкторе

Ранее вы инициализировали поля объекта в конструкторе класса, используя явные аргументы. Однако существует и другая техника, которая основана на применении списка инициализации. Ниже она демонстрируется в альтернативной версии конструктора класса *СВох*.

*// Определение конструктора с использованием списка инициализации*

*CBox():*

*m\_Length(1.0), m\_Width(2.0), m\_Height(1.0)*

*{*

*}*

Способ написания определения этого конструктора предполагает, что он появляется в теле определения класса.

Обратите внимание, что список инициализации для конструктора отделен от списка параметров двоеточием, и каждый инициализатор отделен запятой. Также , список инициализаторов не заканчивается точкой с запятой.

Приватные члены класса

Наличие конструктора, который устанавливает значения полей объекта класса, но при этом открывает возможность любой части программы вмешаться во внутреннее содержимое объекта, по определению противоречиво

Необходимую защиту можно обеспечить, используя ключевое слово *private* при определении членов класса. Члены класса, определенные как *private*, в общем случае могут быть доступны только методам этого же класса. Обычные функции не имеют прямого доступа к *private* членам класса.

Наличие возможности определения членов класса как *private* также позволяет отделить интерфейс класса от его внутренней реализации. Интерфейс класса состоит из общедоступных (*public*) полей и общедоступных методов в частности, поскольку они могут обеспечить непрямой доступ ко всем членам класса, включая *private*. Сохраняя внутренности класса приватными (private), вы можете позднее модифицировать их, без необходимой модификации кода, который использует этот класс через его доступный интерфейс. Чтобы сохранить поля и методы класса защищенными от внешнего вмешательства, правильно объявить их как private. Используйте public с теми полями и методами, которые необходимы для внешнего использования объектов класса.

//listing 41

//Класс с приватными членами

#include <iostream.h>

class CBox //Определение класса в глобальном контексте

{

public:

// Определение конструктора с использованием списка инициализации

СВох (double lv = 1.0, double bv = 1.0, double hv = 1.0) :

*m\_Length(lv), m\_Width(wv), m\_Height(hv)*

{

cout << endl << "constructor work”;

}

//Функция для вычисления объема ящика

double Volume()

{

return m\_Lenght\*m\_Widht\*m\_Height;

}

**private:**

**double m\_Lenght; //Длина ящика в дюймах**

**double m\_Widht; //Ширина ящика в дюймах**

**double m\_Height; //Высота ящика в дюймах**

};

int main ()

{

CBox box2; //объявление box2 без инициализации

CBox match(2.2,1.1,0.5); // объявление match

cout <<endl

<<" V match ="

<< match.Volume(); //Вычислить объем match

//Раскомментировать следующую строку для получения ошибки

// box2. m\_Lenght = 4.0;

cout <<endl

<<" V box2 ="

<< box2.Volume();

cout <<endl;

return 0;

}

Результат

constructor work

constructor work

V match =1.21

V box2 =1

Описание полученных результатов

Определение класса *СВох* теперь состоит из двух разделов. Первый - *public* - содержит конструктор и метод *Volume ()*. Второй раздел определен как *private* и содержит поля. Теперь они могут быть доступны только для методов класса. Не нужно изменять методы - они по-прежнему могут обращаться ко всем полям класса. Но если вы удалите комментарий в *main ()* с оператора, который присваивает значение полю *m\_Length* объекта *bох2*, то получите ошибку компиляции, которая сообщит, что член данных не доступен. Если вы еще этого не сделали, обратите внимание на члены класса *СВох* в Class View (Представление классов). Там вы увидите рядом с каждым членом пиктограмму, указывающую его доступность; маленький замочек на пиктограмме означает, что член является приватным (private).

Важно помнить, что теперь использование конструктора или метода класса - это единственный способ получить значение полей *private* объекта. Вы должны удостовериться, что все способы, которыми вы пожелаете устанавливать или моди­фицировать поля *private* класса, представлены через методы класса.

В раздел *private* класса можно также помещать методы. В этом случае они могут вызываться только из других методов того же класса. Если вы поместите метод *Volume ()* в раздел *private*, то получите сообщение компилятора об ошибке в операторе, который пытается вызвать этот метод внутри функции *main ().*Если в раздел *private* поместить конструктор, то вы не сможете объявить ни одного объекта типа этого класса.

Результат демонстрирует, что класс по-прежнему работает удовлетворительно, хотя его поля определены с атрибутом доступа *private*. Основное отличие в том, что теперь они полностью защищены от неавторизованного доступа и модификации.

Если вы не укажете иначе, атрибутом доступа по умолчанию, применяемым ко всем членам класса, будет *privatе*. Таким образом, вы можете поместить все приватные члены в начало определения класса и позволить им по умолчанию быть

приватными, пропустив ключевое слово доступа. Однако лучше потрудиться и явно определить состояние атрибута доступа в каждом случае, дабы не возникало никаких сомнений относительно ваших намерений.

Доступ к приватным членам класса

Если подумать, объявление полей класса как *private* довольно-таки экс­тремально. Это очень хорошо защищает их от неавторизованных модификаций, но нет причин всегда держать их значение в секрете.

Все, что потребуется сделать - это написать метод, который будет возвращать зна­чение поля. Взгляните на следующий метод класса СВох:

*inline double СВох::GetLength()*

*{*

*return m\_Length;*

*}*

Только для того, чтобы показать, как это будет выглядеть, определение этого метода вынесено за пределы определения класса. Он определен как *inline*, поскольку это дает преимущество в скорости без слишком большого увеличения размера кода. Предполагая, что объявление этой функции находится в разделе public класса, вы можете применять ее в операторах вроде следующего:

*int len = box2 .GetLength ();// Получить член данных, описывающий длину*

Все, что необходимо сделать - написать подобную функцию для каждого поля данных, который вы хотите сделать доступным для чтения внешнему миру, и их значения будут доступны без нарушения безопасности объекта класса. Конечно, если вы поместите определения этих функций внутрь определения класса, они будут встроенными по умолчанию.

Дружественные функции класса

Бывают случаи, когда по той или иной причине , вы хотите некоторым функциям, не являющимся членами класса, разрешить доступ ко всем членам класса, то есть определить разновидность элитной группы со специальными привилегиями.

Такие функции называются дружественными функциями класса и определяются с помощью ключевого слова *friend*. Можно включить либо только прототип дружественной функции в определение класса, либо все определение функции целиком.

Функции, являющиеся друзьями класса и определенные внутри определения класса, так же по умолчанию являются встроенными.

Дружественные функции не являются членами класса – это просто глобальные функции со специальными привилегиями. Атрибуты доступа к ней не относятся.

Предположим, что необходимо реализовать дружественную функцию в классе *СВох* для вычисления площади поверхности объекта *СВох*.

Использование дружественной функции для вычисления площади поверхности

//listing 42

//Использование дружественной функции для вычисления площади поверхности

//Создание дружественной функции

#include <iostream.h>

class CBox //Определение класса в глобальном контексте

{

public:

//определение конструктора

CBox (double lv =1.0, double bv =1.0, double hv =1.0)

{

cout <<endl << "constructor work";

m\_Length = lv; //установить значения полей

m\_Width = bv;

m\_Height = hv;

}

//Функция для вычисления объема ящика

double Volume()

{

return m\_Length\*m\_Width\*m\_Height;

}

private:

double m\_Length; //Длина ящика в дюймах

double m\_Width; //Ширина ящика в дюймах

double m\_Height; //Высота ящика в дюймах

**//Дружественная функция**

**friend double BoxSurface (CBox aBox);**

};

**//Дружественная функция для вычисления площади поверхности объекта CBox**

**double BoxSurface (CBox aBox)**

**{**

**return 2.0\*(aBox.m\_Length \* aBox.m\_Width +**

**aBox.m\_Length \* aBox.m\_Height +**

**aBox.m\_Height \* aBox.m\_Width);**

**}**

int main ()

{

CBox match (2.2,1.1,0.5); //объявление match типа CBox

CBox box2; //объявление box2- без иницализирующих значений

cout <<endl

<<"V match =" <<match.Volume();

cout <<endl << " Surface match =" << **BoxSurface (match)**;

cout <<endl

<<" V box2 ="

<<box2.Volume(); //Вычислить объем Box2

cout <<endl <<"Surface box2 =" << **BoxSurface (box2)**;

cout <<endl;

return 0;

}

Результат

constructor work

constructor work

V match =1.21

Surface match =8.14

V box2 =1

Surface box2 =6

Описание полученных результатов

Вы объявляете функцию *BoxSurfасе ()* как друга класса *СВох*, написав ее прототип с ключевым словом *friend* в начале. Поскольку сама функция *BoxSurfасе ()* является глобальной, неважно, куда именно вы поместите объявление *friend* внутри объявления класса, но будет правильно придерживаться определенного стиля в размещении этого объявления. Желательно поместить ее после всех *public*- и *private*-членов класса. Помните, что дружественная функция не является членом класса, поэтому атрибуты доступа к ней не относятся.

Определение функции следует за классом. Обратите внимание, что вы определяете доступ к полям объекта внутри определения *BoxSurfасе ()*, используя объект *СВох*, переданный функции в качестве параметра. Поскольку дружественная функция не является членом класса, она не может ссылаться на поля класса просто по именам; они должны быть квалифицированы именем объекта - точно так же, как в обычной функции, за исключением того, конечно, что обычная функция не может получить доступ к приватным членам класса.

Дружественная функция - это тоже самое, что и обычная функция, но с тем отличием, что ей разрешен доступ ко всем членам класса или классов, другом которых она является, безо всяких ограничений.

Такого результата следовало ожидать. Дружественная функция вычисляет площадь поверхности объектов СВох по значениям приватных членов.