выкл.

Далее

Суть ООП + 3 осн. понятия.	Аксессоры	Конструктор
Деструктор	Делегирование	Статические члены класса
Конструктор копирования	Указатель this	Инициализаторы
Константные методы	Перегрузка операторов	Побитовое копирование
Преобразование типов ---explicit---	Виды перегрузок	Дружественные функции

Назад

выкл.

Далее

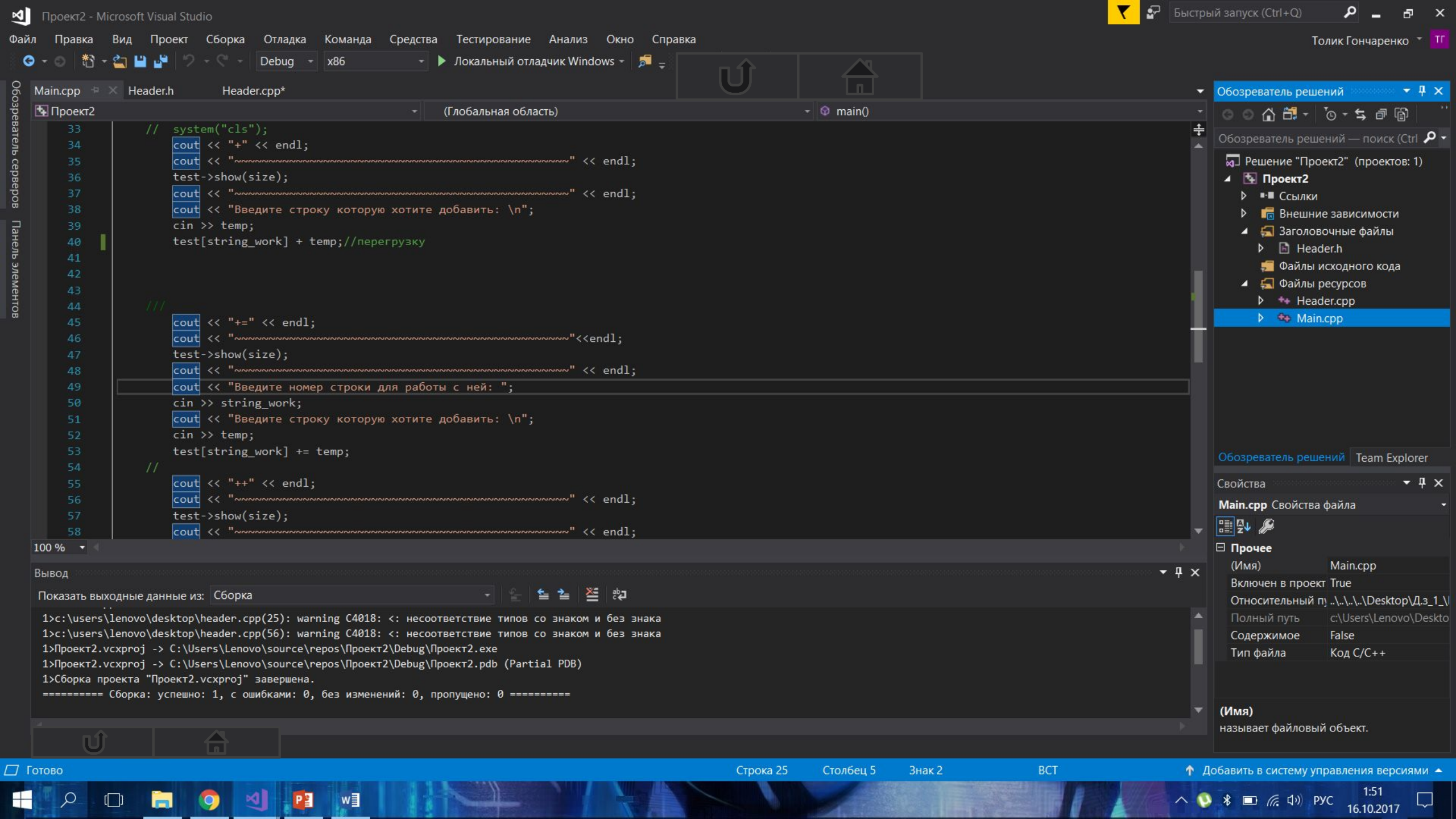
Назад

выкл.

Далее

Назад

выкл.


Проект2 - Microsoft Visual Studio
Быстрый запуск (Ctrl+Q)
Файл Правка Вид Проект Сборка Отладка Команда Средства Тестирование Анализ Окно Справка
Толик Гончаренко
Debug
x86
Локальный отладчик Windows
Main.cpp
Header.h
Header.cpp*
Проект2
(Глобальная область)
main()
33 // system("cls");
34 cout << "+" << endl;
35 cout << "~~~~~~" << endl;
36 test->show(size);
37 cout << "~~~~~~" << endl;
38 cout << "Введите строку которую хотите добавить: \n";
39 cin >> temp;
40 test[string_work] + temp;//перегрузку
41
42
43
44 ///
45 cout << "+=" << endl;
46 cout << "~~~~~~"<<endl;
47 test->show(size);
48 cout << "~~~~~~" << endl;
49 cout << "Введите номер строки для работы с ней: ";
50 cin >> string_work;
51 cout << "Введите строку которую хотите добавить: \n";
52 cin >> temp;
53 test[string_work] += temp;
54 //
55 cout << "++" << endl;
56 cout << "~~~~~~" << endl;
57 test->show(size);
58 cout << "~~~~~~" << endl;
100 %
Вывод
Показать выходные данные из: Сборка
1>c:\users\lenovo\desktop\header.cpp(25): warning C4018: <: несоответствие типов со знаком и без знака
1>c:\users\lenovo\desktop\header.cpp(56): warning C4018: <: несоответствие типов со знаком и без знака
1>Проект2.vcxproj -> C:\Users\Lenovo\source\repos\Проект2\Debug\Проект2.exe
1>Проект2.vcxproj -> C:\Users\Lenovo\source\repos\Проект2\Debug\Проект2.pdb (Partial PDB)
1>Сборка проекта "Проект2.vcxproj" завершена.
========== Сборка: успешно: 1, с ошибками: 0, без изменений: 0, пропущено: 0 ==========
Обозреватель решений
Обозреватель решений — поиск (Ctrl
Решение "Проект2" (проектов: 1)
Проект2
Ссылки
Внешние зависимости
Заголовочные файлы
Header.h
Файлы исходного кода
Файлы ресурсов
Header.cpp
Main.cpp
Обозреватель решений Team Explorer
Свойства
Main.cpp Свойства файла
Прочее
(Имя) Main.cpp
Включен в проект True
Относительный п ..\..\..\..\Desktop\Д.з_1_\
Полный путь c:\Users\Lenovo\Deskto
Содержимое False
Тип файла Код C/C++
(Имя)
называет файловый объект.
Готово
Строка 25
Столбец 5
Знак 2
ВСТ
Добавить в систему управления версиями
РУС
1:51
16.10.2017

Суть ООП заключается в анализе предметной области будущего проекта и разбиении ее на завершенные сущности.

Все оop базируется на 3 основных понятиях:
1.инкапсуляция,
2.наследование,
3.полиморфизм.

инкапсуляция - механизм базирующийся на принципе отделения реализации от интерфейса путем сокрытия данных по сути абстрагирование от реализации.

наследование - механизм предназначенный для организации одного из способов повторного использования кода с целью абстрагирования от типов.

полиморфизм - механизм позволяющий одному интерфейсу иметь множество реализаций.

интерфейс - средство взаимодействия с пользователем.

Правила доступа можно настраивать путем использования спецификаторов доступа.
существует 3 таких спецификатора :
1. **public** - доступ к членам открыт из любой точки программы через объект класса.
2. **private** - доступ к членам возможен только в рамках методов данного класса.
3. **protected** - доступ к членам возможен только в рамках методов данного класса и в рамках методов классов от него унаследованных.

Физическим проявлением инкапсуляции является **сущность**.

Сущность состоит из набора характеристик ее описывающих и набора действий которые можно произвести над этими характеристиками.

Для получения доступа к полям класса принято создавать набор так называемых **аксессоров** - методы, которые позволяют читать и записывать данные в поля (1 метод == 1 поле).

Методы на **чтение** называются:
геттер/гетчер/инспекторы,

Методы на **запись** называются:
сетчеры/сеттеры/модификаторы

Распространенной практикой является вынос описания методов за тело класса, при этом следует помнить что методы, описание которых осталось в классе по умолчанию получают спецификатор **inline**

Сигнатура чтение:
тип поля get Название Поля()
{
return поле;
}

Сигнатура запись:
тип_поля set НазваниеПоля(вход.пар.)
{
тело;
}

Конструктор это метод который нужен для инициализации объекта класса в момент его создания, срабатывает автоматически и имеет специфический синтаксис:
- Такое же название как у класса.
- Отсутствие какого-либо возвращаемого значения даже ***void.***
- В одном классе может быть несколько конструкторов (т.е. конструкторы могут быть перегруженными).
- Если в классе не объявить ни одного конструктора то компилятор создаст так называемый конструктор по умолчанию, у него будет пустое тело и он не будет принимать параметры, но факт его наличия должен быть.
- Конструктор без параметров тоже порой называют конструктором по умолчанию
если есть **хотя бы один** конструктор вне зависимости от того принимает он параметры или нет, компилятор уже не создаст свою версию.

```
// Любой метод может быть
// вынесен  за тело класса  и более
// того  перенесен в другой файл, в
// этом случае в теле класса остается
// прототип метода, а за его
// пределами метод описывается
// следующим образом :

[тип] Имя_класса :: Имя_метода (список параметров)
{
реализация метода;
}
```

выкл.

меню

Деструктор

Деструктор это специальный метод класса, автоматически вызываемый в момент уничтожения объекта.

Деструктор никогда не принимает параметров.
Не возвращает значений.
Если в классе нет деструктора, компилятор создаст его по умолчанию с пустым телом.

объекты всегда уничтожаются в порядке обратном к их созданию, как правило для локальных объектов.

```
~Имя_класса ( )
{
тело
}
```

Имя_класса имя_объекта = Имя_класса(параметры);

Имя_Класса имя_объекта (параметры)

Делегирование в общепринятом бытовом понимании заключается в использовании одним объектом другого с целью решения той или иной задачи которую этот другой объект уже умеет решать.

Делегирование конструкторов это процесс (механизм), при котором один конструктор делегирует (перенаправляет) часть возложенной на него задачи другим конструкторам умеющим эту задачу решать
синтаксис делегирование:

```
имя_класса(параметры(формальные)):имя_класса(передача пар)
{
тело
}
```

При делегировании первым всегда срабатывает конструктор которому делегировали задачу и только потом тот кто делегировал задачу

Статическим полем класса называют поле объявленным с ключевым слова static, данное поле является общим для всех объектов класса.
Статический метод это метод объявленный как член класса со спецификатором **static,** этот метод вызывается через Имя класса т.е без необходимости создания и использовании объекта, этот метод не видит обычные поля и методы класса напрямую.
Такой вид методов применяется для работы со статическими членами класса.
Особенности синтаксиса при работе со статическими членами, статическое поле создается внутри класса с ключевым словом static а инициализируется за его пределами, обязательно в файле реализации чтоб избежать повторного переопределения:

Тип имя_класса::имя_поля = знач

Обращение к статическому полю класса в обычном методе ничем не отличается от обращения к обычному полю.

Конструктор копирование и необходимость его применения

Конструктор копирования это побитовое копирование.
Суть **побитового копирования** заключается в том что создается точная копия некоторого объекта но только объекта на **динамическую** память выделенную под его поля данный процесс никак не влияет, проблема заключается в том что из-за побитового копирования можно получить 2 и более объектов связанных с одной областью в куче.
Существует 4 случая возникновения побитового копирования
1. Инициализация объекта при создании другим уже существующим объектом.
2. Передача объектов в функцию или метод по значению.
3. Возврат объекта из функции или метода по значению // при таком способе возврата //создается ВРЕМЕННЫЙ //объект, являющийся точной //копией возвращаемого.
4. Использование оператора присваивания для 2 объектов одного типа.

Общий синтаксис конструктора копирования

Имя_класса (const Имя_класса & имя_объекта)

Наличие ссылки обусловлено тем что передача объекта по ЗНАЧЕНИЮ спровоцирует ЦИКЛИЧЕСКИЙ вызов конструктора копирования, ссылка константная так как требуется запретить изменение объекта с которого снимается копия

Указатель this это константный указатель передает адрес объекта который передает метод.

Как известно не статические методы класса имеют прямой доступ ко всем членам этого класса вне зависимости от их уровня доступа и местоположения, данная особенность объясняется тем что любой нестатический метода класса на входе всегда принимает неявный указатель на данный класс, этот указатель является константным,
в момент вызова не статического метода класса в данный указатель передается адрес того объекта который вызвал метод.
При обращению внутри метода к конкретным членам класса на самом деле происходит неявное разыменование данного указателя

Такой указатель не передается в СТАТИЧЕСКИЕ методы

```
class Point{                    class Point{
private:                        private:

int x=32;                       int x=32;
int y=47;                       int y=47;
public:                         public:
void show(){                    void Show(Point&const this)
cout<<x<<' '<<y<<'\n';          cout<<this->x<<' '<<this->y<<'\n'
}                               }
}
Point z;                        Point z;
z.show();                       z.show(z);
```

Указатель this при необходимости можно использовать явно

Инициализатор это механизм который срабатывает после создания объекта но до выполнения тела конструктора предназначен для инициализации полей.

Основные причины использования:
1. В классе присутствует объект другого класса у которого отсутствует конструктор без параметров.
2. В классе в качестве поля используется константа которая для каждого объекта должна иметь некое уникальное неизменяемое значение но свое.
3. В классе в качестве поля используется ссылку которую нужно инициализировать при создании.

Если класс разбивается на класс дизайна и файл реализации то инициализаторы должны быть описаны в файле реализации.

Общий синтаксис использования инициализаторов
ИмяКласса([пар1],[пар2]):поле1(пар1),поле2(пар2), поле3(знач){тело;}

Константный метод это метод который гарантированно ни прямо ни косвенно модифицировать поля класса.

Данный вид методов имеет 2 предназначения :
1. Контроль за действиями разработчика.
2. Через константные сущности(объекты, ссылки, указатель на константы) можно вызвать только константные методы.

Ключевое слово **mutable** указанное перед конкретным полем класса говорит о том что это поле имеет полное право меняться где захочет в т.ч. и в константных методах.

В одном классе может быть 2 метода с одинаковой сигнатурой но при этом один из них будет **const** а другой нет…
В этой ситуации однозначность вызова определяется типом вызывающего объекта и типом не явно переданного в метод указателя this, если вызывающий объект является константой то указатель this представляет собой константный указатель на константу, если вызывающий объект переменный(изменяемый), то тогда указатель **this** является просто константным указателем, например:
Point*const this
Point const*const this

Статические методы не могут быть константными
тип имя()const{ тело;}
спецификатор const нужно указывать в обоих файлах…

Перегрузка операторов это механизм перегрузки операторов.

Если в программе необходимо организовать работу в стиле префиксной и постфиксной формы инкремента или декремента то тогда следует учитывать что при реализации:
1. Префиксной формы необходимо изменить состояние объекта и этот измененный объект подставить на место вызова.
2. Постфиксной формы необходимо изменить состояние объекта но на место вызова подставить объект находящийся в старом состоянии.

Когда мы делаем перегрузку бинарного оператора используя средства (класс == методы класса), мы должны передавать в него всего один параметр который представляет собой объект находящийся справа от оператора, левым операндом будет в этом случае выступать объект класса который инициировал вызов метода.
Фиктивный параметр нигде не учитывается и не используется, он нужен только для отличия форм.

Тернарный оператор перегрузить невозможно.

Побитовое копирование это конструктор копирования.

Оператор **“=”** обладает интересным свойством транзитивности который заключается в возможности множественного присваивания объектов друг другу.
Например выражение A=B=C=D=E.

Перегрузка оператора **“=”** похожа на реализацию конструктора копирования фактически есть только 3 отличия:
1. Необходимо очищать память выделенную под объект располагающуюся слева от оператора = .
2. Необходимо возвращать объект в его новом измененном состоянии.
3. Необходимо осуществлять проверку на копирования объекта в себя.

Довольно часто если внутри класса скрывается коллекция элементов необходимо получать доступ извне к этим элементам по отдельности, вполне очевидным и естественным является использование оператора индексирования **“[]”**. Например:
```
Array A;
cout<<A[3];
A[3]=10;
```
Вторым достаточно интересным специальным оператором является оператор **“()”** в классическом своем применении данный оператор предназначен для передачи списка параметров в какую-либо функцию или метод, однако механизм перегрузки позволяет использовать его по отношению уже к существующему объекту например для инициализации каких-либо полей или членов особенность оператора заключается в том что кол-во и типы принимаемых им параметров могут быть любыми как и возвращаемое значение.
Синтаксис:
```
тип_возвр operator()([список параметров]){}
```

Таких операторов в классе может быть много главное чтоб они отличались параметрами по принципу перегрузки функций и характеризовались однозначностью вызова.

Варианты преобразования:
1. наш_польз_тип 1->наш
2. наш_польз_тип1->чужой
3. наш -> стандратный
4. чужой -> наш
5. стандарт -> наш

Примечание - если классы локализуются в нескольких разных файлах и один класс использует ***объект*** другого класса, то об этом другом классе должно быть известно все , поэтому его h файл должен быть подключен до использования (создания) объекта.

Если где-либо используется исключительно упоминание типа (возвр значение, ссылка, указатель, оператор преобразования) то тогда достаточно просто сообщить компилятору что такой класс в последствии потенциально будет существовать, это можно сделать с помощью конструкции под названием прототип класса например:

class Circle;
Array Temp=3;

В данном случае для объекта Temp сработает конструктор который в состоянии принять на входе один параметр типа int такое преобразование носит название **преобразование с помощью конструктора.**

Среди прочих операторов существует так называемый оператор типа и этот оператор тоже можно перегружать, общий синтаксис этого оператора следующий :

operator тип_к_котором_приводим();
operator типа();

Explicit нужен чтобы запретить преобразование типов:

Explicit Point (int t=0)
Point a=5;

Виды перегрузок:

Перегрузка методов
- operator
- бинарные операторы 1 -пар
- унарные - 0пар
- если наш тип стоит слева
- public
- нестатические

Глобальная перегрузка
(за счёт функции)
- operator
- бинарные операторы 2 -пар
- унарные - 1 пар
- наш тип где угодно

Дружественные функции

Существует 3 вида дружбы:
1. Функция дружит с классом (в этом случае функция просто получает доступ к приватным и защищенным членам класса через объект).
2. Метод одного класса дружит с другим классом (метод будет видеть через объект защищенные и закрытые поля).
3. Класс дружит с классом (в этом случае все методы дружественного класса становятся дружественными по отношению ко 2 классу).

Дружба должна быть подтверждена внутри того кто предоставляет свое приватное устройство, для обычной функции синтаксис френд и его прототип.

Перегрузка ввода, вывода:
Ввод и вывод в стиле c++ осуществляется средствами 2-х специальных объектов связанных с выходным и входным потоком в консоли, эти объекты созданы в рамках библиотеки iostream, там же проинициализированы и привязаны к потоку в качестве интерфейса для ввода вывода библиотека предоставляет перегруженные для различных типов операторы побитового сдвига (<< >>), в рамках библиотеки iostream объект cin имеет тип данных istream a объект cout имеет тип данных ostream.

В istream реализованный побитовый сдвиг вправо а в ostream побитовый сдвиг влево.