**Дружественная функция** — это функция, которая имеет доступ к закрытым членам класса, как если бы она сама была членом этого класса. Во всех других отношениях дружественная функция является обычной функцией. Ею может быть, как обычная функция, так и метод другого класса. Для объявления дружественной функции используется **ключевое слово** **friend** перед [**прототипом функции**](https://ravesli.com/urok-19-predvaritelnoe-obyavlenie-i-prototip-funktsii/), которую вы хотите сделать дружественной классу. Неважно, объявляете ли вы её в public- или в private-зоне класса. Например:

class Anything

{

private:

    int m\_value;

public:

    Anything() { m\_value = 0; }

    void add(int value) { m\_value += value; }

    // Делаем функцию reset() дружественной классу Anything

    friend void reset(Anything &anything);

};

// Функция reset() теперь является другом класса Anything

void reset(Anything &anything)

{

    // И мы имеем доступ к закрытым членам объектов класса Anything

    anything.m\_value = 0;

}

int main()

{

    Anything one;

    one.add(4); // добавляем 4 к m\_value

    reset(one); // сбрасываем m\_value в 0

}

Здесь мы объявили функцию reset(), которая принимает объект класса Anything и устанавливает m\_value значение 0. Поскольку reset() не является членом класса Anything, то в обычной ситуации функция reset() не имела бы доступ к закрытым членам Anything. Однако, поскольку эта функция является дружественной классу Anything, она имеет доступ к закрытым членам Anything.

Обратите внимание, мы должны передавать объект Anything в функцию reset() в качестве параметра. Это связано с тем, что функция reset() не является методом класса. Она не имеет [**указателя \*this**](https://ravesli.com/urok-121-skrytyj-ukazatel-this/) и, кроме как передачи объекта, она не сможет взаимодействовать с классом.

**Дружественные классы**

Один класс может быть дружественным другому классу. Это откроет всем членам первого класса доступ к закрытым членам второго класса, например:

#include <iostream>

class Values

{

private:

    int m\_intValue;

    double m\_dValue;

public:

    Values(int intValue, double dValue)

    {

        m\_intValue = intValue;

        m\_dValue = dValue;

    }

    // Делаем класс Display другом класса Values

    friend class Display;

};

class Display

{

private:

    bool m\_displayIntFirst;

public:

    Display(bool displayIntFirst) { m\_displayIntFirst = displayIntFirst; }

    void displayItem(Values &value)

    {

        if (m\_displayIntFirst)

            std::cout << value.m\_intValue << " " << value.m\_dValue << '\n';

        else // или сначала выводим double

            std::cout << value.m\_dValue << " " << value.m\_intValue << '\n';

    }

};

int main()

{

    Values value(7, 8.4);

    Display display(false);

    display.displayItem(value);

}

Поскольку класс Display является другом класса Values, то любой из членов Display имеет доступ к private-членам Values.

**Примечания о дружественных классах:**

   Во-первых, даже несмотря на то, что Display является другом Values, Display не имеет прямой доступ к указателю \*this объектов Values.

   Во-вторых, даже если Display является другом Values, это не означает, что Values также является другом Display. Если вы хотите сделать оба класса дружественными, то каждый из них должен указать в качестве друга противоположный класс. Наконец, если класс A является другом B, а B является другом C, то это не означает, что A является другом C.

Будьте внимательны при использовании дружественных функций и классов, поскольку это может нарушать принципы [**инкапсуляции**](https://ravesli.com/urok-115-inkapsulyatsiya-gettery-i-settery/). Если детали одного класса изменятся, то детали класса-друга также будут вынуждены измениться. Следовательно, ограничивайте количество и использование дружественных функций и классов.