Т.2.7.4. Друзья классов.

Дружественная функция

Дружественная функция — это функция, которая не является членом класса, но имеет доступ к членам класса, объявленным в полях private или protected.

- Дружественные функции должны определяться вне класса, для которого они объявляются друзьями, а об особых отношениях между ними и данным классом свидетельствует лишь специальное объявление(!) со **спецификатором объявления friend.**
- Дружественная функция **может располагаться в любом поле класса** – private, public или protected.
- При определении дружественной функции, **элементы класса необходимо явно передавать в нее в виде параметров функции.**
- В виде параметра, в дружественную функцию **надо передать указатель или ссылку на объект класса.** Иначе она не увидит данные какого класса ей принять и обработать.
- Функция может использоваться, как дружественная к нескольким классам.

Дружественные функции

- описания **friend** не взаимны: если A объявляет B другом, то это не означает, что A является другом для B;
- дружественность не наследуется: если A объявляет B другом, классы, производные от B, не будут автоматически получать доступ к элементам A;
- дружественность не является переходным свойством: если A объявляет B другом, классы, производные от A, не будут автоматически признавать дружественность B.

Задача

Создать класс **Woman25,** который, используя дружественные функции и обычные методы класса, будет получать данные об объекте (имя и вес) и выводить их на экран.

Методы и friend-функции будут выполнять аналогичные действия.

На основании полученных данных, программа даст пользователю совет относительно корректировки его веса.

Задача

```
class Woman25
{ private:
   char *name;   //имя
   int weight;     //вес
   friend void setData(char *, int, Woman25&);  //объявление  др. функций
   friend void getData(Woman25&);
public:
   Woman25()    //конструктор
   {   name = new char [20];
      strcpy(name, "Норма");
      weight = 60;     }
   ~Woman25()//деструктор
   {   delete [] name;
      cout << "Деструктор" << endl;     }
    void setData(char*, int); //объявление методов класса
   void getData();
   void advise();  };
```

Задача

```
void setData(char *n, int w, Woman25& object)   //определяем friend-
    ф
  {
     strcpy(object.name, n);
     object.weight = w;
  }

  void getData(Woman25& object)//определяем friend-функцию
  {
     cout << object.name << "\t: " << object.weight  << " кг" << endl;
  }
```

Задача

```
void Woman25::setData(char *n, int w)  //определяем set-метод
    класса
  {   strcpy(name, n);
      weight = w;    }
   void Woman25::getData()    //определяем get-метод класса
  {       cout << name << "\t: " << weight  << " кг" << endl;         }
   void Woman25::advise()     //определяем метод класса advise
  {
     if(weight < 55)
        cout << "Вам надо потреблять больше калорий!" << endl;
        else if(weight >= 55 && weight <= 65)
        cout << "Ваш вес в норме!" << endl;
         else
        cout << "Вам надо ограничивать себя в еде!" << endl;
  }
```

Задача

```
int main()
{
   Woman25 Norm; //создаем объект Norm
   Norm.getData();
   Woman25 Duna; //второй объект
   Duna.setData(“Дуня", 100);
   Duna.getData();
   Duna.advise();
   Woman25 Inna; //третий объект
   setData("Инна", 50, Inna);
   getData(Inna);
   Inna.advise();
 return 0;
}
```

Вывод:

Представьте, что у нас есть еще десяток классов. Например **Girl6_7, Girl8_9, Man25** и т.д.,

Используя дружественные функции, нам не придется для каждого класса определять set и get-методы.

А так достаточно определить метод в одном из классов или вообще определить функцию, как глобальную, а в остальные классы прописать ее прототип, как дружественной функции (используя слово friend).

Экономится масса времени и код становится намного короче.