

Класс **whatever**

Иногда возникает потребность в хранении разнотипных объектов в одном «общем» типе. Можно, конечно, воспользоваться C-style техникой и хранить все в `void*`, но, вероятнее всего, это приведет к немалому количеству ошибок.

В данной работе предлагается разработать **нешаблонный** класс **whatever**, объект которого может хранить в себе значение произвольного типа (с небольшой оговоркой на требование иметь конструктор копирования). Объект класса **whatever** должен позволять «безопасно» предоставлять значение этого типа по запросу. «Безопасно» в данном случае предполагает, что если **whatever** содержит объект типа `T`, то при запросе:

- типа `T`, значение будет предоставлено;
- типа, отличного от `T`, пользователь класса **whatever** будет проинформирован об ошибке.

Конструирование

Объект класса **whatever** можно создать от объекта произвольного* типа (**`decay<T>`** при этом `copy constructible`, см. ниже) :

```
int a = 5;
whatever wa(a);

string b = 6;
whatever wb(b);
```

Объекты класса **whatever** хранят значение. Именно поэтому они содержат не объект переданного произвольного типа `T`, а его суть, выражаемую через вспомогательную метафункцию **`std::decay()`**. Так ссылки теряют свою константность и “ссылочность”, массивы превращаются в указатели и т.д. Ну а `value type` (как `int` или `string`) остаются `int`’ом и `string`’ом. Просто храните внутри не `T`, а **`std::decay<T>()`**.

Получение значения

Получение значение происходит по аналогии с **`dynamic_cast()`**:

```
template<typename T> const T* whatever_cast(const whatever *);
template<typename T> T* whatever_cast(whatever*);
template<typename T> T whatever_cast(whatever &);
template<typename T> T whatever_cast(const whatever &);
```

Из объекта класса **whatever** можно получить значение, если оно там есть, и тип **std::decay<T>** совпадает с типом, хранимым в этом объекте **whatever**.

Если в **whatever_cast()** передан указатель на **whatever**, и шаблонный тип совпадает с хранимым в объекте, возвращается указатель на хранимый объект (той же константности, что и указатель на **whatever**). Если не совпадает, или внутри **whatever** ничего нет - возвращается нулевой указатель.

При вызове по ссылке от **whatever**, если **T** - ссылка, возвращается ссылка, иначе копия. Константность соответствует константности переданного **whatever**.

Если значение не получить (ничего в **whatever** нет, или тип не тот), тогда бросается исключение **bad_whatever_cast**.

Задание

В пространстве имен **utils** реализуйте нешаблонный класс **whatever**, содержащий:

1. конструктор по умолчанию,
2. конструктор от типа **T** (**decay<T>** при этом `copy constructible`),
3. корректный деструктор (удаляющий внутренний объект, если он там есть).

Поддержите корректное копирование **whatever**: т.е. у **whatever** должны быть согласованные `copy constructor` и `copy assignment operator`. При этом **whatever** копирует внутреннее значение, если это возможно, и исходный **whatever** не пуст.

Также реализуйте:

1. функции **whatever_cast()** от указателя на **whatever** (с константностью и без),
2. **swap** для двух **whatever**,
3. функции **empty()** и **clear()** по аналогии со стандартными контейнерами.

Решение должно быть оформлено в файле **whatever.hpp**.

Подсказки

1. Разрабатываемый класс нешаблонный. При этом принимает произвольный тип. Вывод - конструктор шаблонный.
2. Нешаблонный тип хранит произвольный тип. Да еще и позволяет достать его тогда и только тогда, когда запрашивается именно исходный тип (с точностью до **std::decay**). Как? Да очень просто, используя концепцию стирание типа - **whatever** хранит указатель на интерфейс, за которым прячется шаблонная реализация. Интерфейс (да и реализация) имеют функцию, выдающую **type_info** - так тип и проверяется.