# Лабораторная работа №2

std::stringstream

auto, override, final, delete, default, move, forward

## Задание 1

Требуется создать универсальную «обертку» для:

* хранения данных любого типа
* различных манипуляций с данными любого типа

Например:

{

WrapAny n1(1);

WrapAny n2(2);

WrapAny n3(3);

n1 += n2;

n3 = n1 + n2;

//или

WrapAny d1(1.1);

WrapAny d2(2.2);

WrapAny d3(3.3);

d1 += d2;

d3 = d1 + d2;

//или

WrapAny s1(std::string("AAA"));

WrapAny s2(std::string("BBB"));

WrapAny s3(std::string("CCC"));

s1 += s2;

s3 = s1 + s2;

//или

WrapAny pt1(MyPoint(1,1));

WrapAny pt2(MyPoint(2,2));

WrapAny pt3(MyPoint{ 3,3 });

pt1 += pt2;

pt3 = pt1 + pt2;

//и даже так:

std::vector<WrapAny> v;

v.push\_back(WrapAny(33));

v.push\_back(WrapAny(5.5));

v.push\_back(WrapAny(std::string("aa")));

v.push\_back(WrapAny(MyPoint(7, 7)));

for (const auto& x : v) { std::cout << x<<' '; }

auto it = std::find(v.begin(), v.end(), WrapAny(33));

}

Подсказки:

1. Универсальным (не зависящим от типа) является строковое представление данного => объединить разные типы можно посредством строкового представления => **базовый класс** может содержать строковое представление данного + интерфейс (чисто виртуальные) методы, которые в зависимости от актуального типа должны быть реализованы (естественно по-разному) в каждом производном классе. Например:  
   **1                          - "1"  
   2.34                     - "2.34"  
   Point(5, 10)          - "5 10"  
   std::string("ABC") - "ABC" и т.д.**
2. Базовый класс ничего не знает про своих наследников, но предоставляет любому производному классу строковое представление, который тот по необходимости преобразует в актуальный тип и обратно (это удобно делать посредством класса std::stringstream)  
   Например:  
   std::stringstream stream;  
   stream<<1; //”1”  
   int n;  
   stream>>n; //1  
     
   Замечание: для надежного повторного использования объекта stream:  
   stream.str("") ; //один из способов очистки буфера (в любом случае нужно, чтобы внутренний указатель переместился на начало буфера)  
   stream.clear(); //сброс флагов
3. Базовый класс **НЕ может быть шаблонным**, так как в этом случае объединить в одном хранилище int, double, string, MyPoint, … будет невозможно.   
   Базовый класс – вспомогательный => нельзя позволять создавать объекты базового типа.  
   Нужно подумать о том, как разделить обязанности между базовым классом и классами – потомками. Также возможно понадобится нетрадиционная реализация перегруженных операторов…

class Base\_Str {

<спецификатор\_доступа???>:

std::string m\_str;//строковое представление любого объекта

<спецификатор\_доступа???>:

…

//требуемые методы??? Что можно разрешать, а чего точно нельзя?

//как предусмотреть эффективную передачу временных объектов в качестве параметра конструктора?

//как задавать сигнатуру виртуальных методов??? Если метод возвращает ссылку? Что делать, если метод **по правилам** должен возвращать по значению?

//как осуществлять вывод значения в поток?

…

};

1. Производный класс:

Производный класс не содержит данных (строковое представление данного хранится в универсальной базовой части), но отвечает за совершение всех требуемых операций!

Это **ШАБЛОННЫЙ** класс, так как его функциональность **зависит** от типа объекта!

#include <sstream>

template<typename T> class Any : <спецификатор\_наследования???> Base\_Str {

//данных нет

<спецификатор\_доступа???>:

//в качестве подсказки – реализация конструктора

Any(const T& t = T())

{

//Перевод параметра в строковое представление и сохранение в базовой части

std::stringstream stream;

stream << t; //что должно быть реализовано для пользовательских типов?

m\_str=stream.str();

}

//Неплохо предусмотреть прием временного объекта в качестве параметра конструктора. А, может быть, стоит объединить два конструктора в один???

…

};

1. Теперь можно объединить объекты производного типа посредством указателей базового типа - Base\_Str\*.   
   Но! «голыми» указателями пользоваться неудобно и опасно => “заворачиваем” такой указатель в класс-обертку – класс WrapAny. Этот класс тоже **не может быть шаблонным**, зато у него может быть **шаблонный конструктор и шаблонные методы**!

class WrapAny{

Base\_Str\* m\_p;

…

public:

//подсказка:

template<typename T> WrapAny(const T& t) { m\_p = new Any<T>(t); }

…

};

Base\_Str\*

string m\_str;

Данных нет!

Операции с

конкретным

типом

WrapAny

Any

Base\_Str

1. Осталось реализовать все необходимые методы, начиная с класса обертки WrapAny:
   1. распечатать значение
   2. operator+
   3. operator+=
   4. operator++
   5. …  
      При этом: операции нужно «делегировать» целевому классу Any посредством указателей или ссылок базового типа Base\_Str
2. При реализации методов класса Any нужно предусмотреть ситуации, когда в качестве параметра оператор получает ссылку на объект другого типа.

## Задание 2. Реализуйте шаблон класса MyUniquePTR, который является оберткой для указателя на объект любого типа.

Задача – класс должен обеспечивать единоличное владение динамически создаваемым объектом. Например для класса A:

class A{

//данные

<спецификатор\_доступа>:

//методы, для безопасного выполнения задачи

public:

void f();

…

};

{

MyUniquePTR<A> p1(new A(<параметры>));

p1->f();

A a = \*p1;

MyUniquePTR<A> p2=p1;

MyUniquePTR<A> p3(new A(<параметры>));

p3 = p2;

vector< MyUniquePTR<A>> v; //как проинициализировать???

list< MyUniquePTR<A>> l;

//как скопировать из v в l ???

}