

Семинар #4: Шаблоны. Домашнее задание.

Задача 1. Шаблонный куб

Напишите шаблонную функцию `cube`, которая будет принимать число любого типа и возвращать куб этого числа того же типа. Пример работы с такой функцией:

```
auto a = cube(5);
std::cout << a << " " << sizeof(a) << std::endl; // Должен напечатать 125 4

auto b = cube(5.0);
std::cout << b << " " << sizeof(b) << std::endl; // Должен напечатать 125.0 8

char x = 5;
auto c = cube(x);
std::cout << c << " " << sizeof(c) << std::endl; // Должен напечатать 125 1
```

Задача 2. Утроение

Пусть была написана шаблонная функция `triple`, которая может принять на вход число любого типа и вернуть это число, умноженное на 3. Функция работает с числами, но нам бы хотелось, чтобы эта же шаблонная функция работала и со строками типа `std::string`. Измените код программы так, чтобы можно было использовать функцию `triple` для объектов типа `std::string`.

```
#include <iostream>

template<typename T>
T triple(const T& x)
{
    return 3 * x;
}

int main()
{
    int a = 10;
    std::cout << triple(a) << std::endl; // Сработает, напечатает 30

    std::string b = "Cat";
    std::cout << triple(b) << std::endl; // Ошибка, нельзя число 3 умножать на std::string
                                         // Нужно чтобы напечаталось CatCatCat
}
```

Решите эту задачу четырьмя способами:

- Измените шаблонную функцию `triple` так, чтобы она работала со строками тоже (вместо умножения используйте сложение).
- Используя перегрузку операторов, напишите `std::string operator*(int n, const std::string& s)`. После этого строки `std::string` могут быть использованы в шаблонной функции `triple`.
- Используйте перегрузку функций и напишите отдельную перегрузку для функции `triple`, принимающую объект типа `std::string`.
- Используйте полную специализацию и напишите специализацию шаблона для типа `std::string`. Если в коде будет одновременно реализована и перегрузка, и специализация, то какая версия выберется?

Задача 3. Минимум и максимум в векторе

Напишите шаблонную функцию, которая будет принимать на вход вектор элементов некоторого типа и возвращать пару, содержащую минимальный и максимальный элемент этого вектора. Известно, что вектор содержит хотя бы один элемент.

```
template<typename T>
std::pair<T,T> minmax(const std::vector<T>& v)
```

Протестируйте данную функцию на векторах, содержащих объекты следующих типов: `int`, `std::string` и `std::pair<int, int>`:

```
#include <iostream>
#include <vector>
#include <utility>
// Тут нужно реализовать шаблонную функцию minmax

int main()
{
    std::vector<int> a {60, 10, 40, 80, 30};
    auto am = minmax(a);
    std::cout << am.first << " " << am.second << std::endl; // 10 80

    std::vector<std::string> b {"Cat", "Dog", "Mouse", "Camel", "Wolf"};
    auto bm = minmax(b);
    std::cout << bm.first << " " << bm.second << std::endl; // Camel Wolf

    std::vector<std::pair<int, int>> c {{10, 90}, {30, 10}, {20, 40}, {10, 50}};
    auto cm = minmax(c);
    std::cout << cm.first.first << " " << cm.first.second << std::endl; // 10 50
    std::cout << cm.second.first << " " << cm.second.second << std::endl; // 30 10
}
```

Задача 4. Сравнение количества элементов в контейнере

Напишите шаблонную функцию `hasMoreElements`, которая бы принимала на вход два контейнера и возвращала бы `true`, если количество элементов в первом контейнере больше, чем во втором и `false` иначе. Под контейнером тут понимается `std::vector`, `std::array`, `std::string` или любой другой класс, хранящий набор элементов и имеющий метод `size`.

```
#include <iostream>
#include <vector>
#include <string>
#include <array>
// Тут нужно реализовать шаблонную функцию hasMoreElements

int main()
{
    std::vector<int> a {10, 20, 30, 40, 50};
    std::string b = "Cat";
    std::string c = "Elephant";
    std::array<int, 3> d {10, 20, 30};
    std::cout << hasMoreElements(a, b) << std::endl; // Должно напечатать 1
    std::cout << hasMoreElements(a, c) << std::endl; // Должно напечатать 0
    std::cout << hasMoreElements(a, d) << std::endl; // Должно напечатать 1
}
```

Задача 5. Изменение порядка байт

Напишите шаблонную функцию `swapEndianness`, которая бы меняла порядок байт объекта скалярного типа с Little Endian на Big Endian или наоборот.

```
#include <iostream>

// Тут нужно реализовать шаблонную функцию swapEndianness

int main()
{
    std::cout << std::hex;

    int a = 0x1a2b3c4d;
    std::cout << a << std::endl; // Должен напечатать 1a2b3c4d
    swapEndianness(a);
    std::cout << a << std::endl; // Должен напечатать 4d3c2b1a
    swapEndianness(a);
    std::cout << a << std::endl; // Должен напечатать 1a2b3c4d

    short b = 0x1a2b;
    std::cout << b << std::endl; // Должен напечатать 1a2b
    swapEndianness(b);
    std::cout << b << std::endl; // Должен напечатать 2b1a
}
```

Задача 6. Целые числа для вычисления по модулю

Напишите шаблонный класс `Modular`, который будет представлять собой целые числа с модульной арифметикой. У класса должно быть 2 шаблонных параметра: тип целого числа, который будет использоваться для хранения модульного числа, и сам модуль. Напишите следующие методы:

1. Конструктор от целого числа.
2. Конструктор копирования.
3. Оператор присваивания от такого же типа.
4. Перегруженные бинарные операторы сложения, вычитания, умножения с числами и с объектами такого же типа.
5. Унарный оператор минус.
6. Оператор `<<` с объектом `std::ostream` для вывода на экран.
7. Конструктор от типа `Modular` с другими шаблонными параметрами.

```
Modular<int, 7> a(10);
std::cout << a << std::endl; // Напечатает 3
a = (a + 8) * 4;
std::cout << a << std::endl; // Напечатает 2

Modular<int, 7> b(a);
b = b + 2;
a = a - b;
std::cout << a << std::endl; // Напечатает 5

Modular<short, 3> c(a);
std::cout << c << std::endl; // Напечатает 2
```

Задача 7. Менеджер создания объекта

Напишите шаблонный класс `Manager`, который будет разделять процессы выделения/освобождения памяти и создания/уничтожения объекта. У этого класса должны быть следующие методы:

- Конструктор по умолчанию.
- `allocate()` - будет выделять необходимое количество памяти под объект типа `T` в куче (используйте `std::malloc`).
- `construct(const T& t)` - будет создавать объект типа `T`, используя конструктор копирования, в выделенной памяти. Используйте оператор `placement new`.
- `destruct()` - будет уничтожать объект в выделенной памяти.
- `deallocate()` - будет освобождать выделенную память.
- `get` - будет возвращать ссылку на объект.

Пример использования данного класса:

```
#include <iostream>
#include <string>
using namespace std::string_literals;
// Тут нужно реализовать класс Manager

int main()
{
    Manager<std::string> a;
    a.allocate();

    a.construct("Cats and dogs"s);
    a.get() += " and elephant";
    cout << a.get() << endl; // Должен напечатать Cats and dogs and elephant
    a.destruct();

    a.construct("Sapere Aude"s);
    cout << a.get() << endl; // Должен напечатать Sapere Aude
    a.destruct();

    a.deallocate();
}
```

Задача 8. Реализация упрощенного аналога `std::optional`

Напишите шаблонный класс `Optional<T>`, который мог бы либо хранить объект типа `T`, либо находиться в пустом состоянии. Класс должен поддерживать следующие методы:

1. Конструктор по умолчанию – должен создавать пустой `Optional`.
2. Конструктор от объекта типа `T`.
3. Конструктор копирования (от объекта типа `Optional<T>`).
4. Конструктор от другого `Optional` (объекта типа `Optional<U>`, где `U` отличается от `T`). Работает при том условии, если объект типа `U` может быть конвертирован в объект типа `T`.
5. Оператор присваивания.
6. Оператор присваивания от другого `Optional` (объекта типа `Optional<U>`, где `U` отличается от `T`).

7. Метод `has_value`, возвращает `true`, если `Optional` хранит объект и `false` иначе.
8. Унарный оператор `operator*`, возвращает ссылку на объект. Если `Optional` находится в пустом состоянии, то UB.
9. Перегрузку унарного оператора `operator*`, которая возвращает константную ссылку на объект. Это метод для работы с константными `Optional`. Если `Optional` находится в пустом состоянии, то UB.
10. Метод `value`, возвращает ссылку на объект. Если `Optional` находится в пустом состоянии, то метод должен бросать исключение `BadOptionalAccess`. Специально для этой подзадачи вам нужно написать этот класс исключения.
11. Перегрузку метода `value`, которая возвращает константную ссылку на объект. Это метод для работы с константными `Optional`. Если `Optional` находится в пустом состоянии, то метод должен бросать исключение `BadOptionalAccess`.
12. Оператор преобразования `explicit operator bool()`. Для того, чтобы можно было использовать `Optional` в условии `if`.
13. Метод `reset`, корректно уничтожает содержимое и переводит `Optional` в нулевое состояние.
14. Метод `value_or`, который принимает значение по умолчанию по константной ссылке. Если в данном `Optional` хранится значение, то `value_or` должен возвращать это значение. Если же данный `Optional` является пустым, то должна возвращаться копия значения по умолчанию.
15. Деструктор.

Для создания/удаления объекта внутри `Optional` при каждом изменении состояния используйте `placement new` / явный вызов деструктора.

Нужно написать упрощённую реализацию без использования `move`-семантики/`rvalue`-ссылок, `type_traits`, спецификаторов `noexcept` и `constexpr`.

Протестируйте вашу реализацию на тестах из файла `seminar4_templates/homework/test_optional.cpp`.

Необязательные задачи (не входят в ДЗ, никак не учитываются)

Задача 1. Простые делители

Напишите функцию `std::vector<std::pair<int, int>> factorization(int n)`, которая будет находить все простые делители числа `n` и их количества.

аргумент	возвращаемое значение
60	{{2, 2}, {3, 1}, {5, 1}}
626215995	{{3, 3}, {5, 1}, {17, 1}, {29, 1}, {97, 2}}
107	{{107, 1}}
1	{{1, 1}}

Задача 2. Указатель или ссылка?

Напишите шаблонный класс `Ref<T>` который будет совмещать свойства указателя и ссылки. Как и указатель этот объект можно будет копировать и положить в контейнеры. Но инициализироваться данный объект должен как ссылка и все операторы, применяемые к этому объекту, должны применяться к тому объекту, на который он ссылается. Правда, к сожалению, перегрузить оператор точка (пока?) нельзя, поэтому вместо этого будем использовать оператор `->`.

- Конструктор от объекта типа `T`.
- Конструктор копирования.
- Оператор присваивания. Присваивание должно производиться к объекту, на который ссылается `Ref`.
- Оператор `+=`.
- Оператор `+`. Должен возвращать новый объект типа `T`.
- Перегруженный оператор `->`
- Функция `get`. Должна возвращать ссылку на объект, на который `Ref` ссылается.
- Дружественный оператор `operator<<(std::ostream&, Ref<T>)`.

Код для тестирования:

```
void toUpper(Ref<std::string> r)
{
    for (size_t i = 0; i < r->size(); ++i)
        r.get()[i] = toupper(r.get()[i]);
}

int main()
{
    int a = 10;
    Ref<int> ra = a;
    ra += 10;
    cout << a << " " << ra << endl;

    std::string s = "Cat";
    Ref<std::string> rs = s;
    rs = "Mouse";
    rs += "Elephant";
    cout << rs << endl;
    cout << s << endl;

    toUpper(s);
```

```

cout << s << endl;

std::vector<std::string> animals {"Cat", "Dog", "Elephant", "Worm"};
std::vector<Ref<std::string>> refs {animals.begin(), animals.end()};

for (int i = 0; i < refs.size(); ++i)
    refs[i] += "s";

for (int i = 0; i < animals.size(); ++i)
    cout << animals[i] << " ";
cout << endl;
}

```

Этот код должен напечатать:

```

20 20
MouseElephant
MouseElephant
MOUSEELEPHANT
Cats Dogs Elephants Worms

```