

## Семинар #4: Шаблоны. Класс `std::vector`. Домашнее задание.

### Задача 1. Сумма чётных

Напишите функцию `sumEven`, которая будет принимать вектор чисел типа `int` по константной ссылке и возвращать сумму всех чётных чисел этого вектора.

аргумент	возвращаемое значение
{4, 8, 15, 16, 23, 42}	70

### Задача 2. Последние цифры

Напишите функцию `lastDigits` которая должны будут принимать на вход вектор чисел типа `int` по ссылке и заменять все его элементы на последние цифры этих элементов.

### Задача 3. 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 4. Времена из строки

- Напишите простой класс для работы со временем:

```
class Time
{
private:
    int mHours, mMinutes, mSeconds;
public:
    Time(int hours, int minutes, int seconds);
    Time(const std::string& s); // строка в формате "hh:mm:ss"
    Time operator+(Time b) const;
    int hours() const; int minutes() const; int seconds() const;
    friend std::operator<<(std::ostream& out, Time t);
};
```

- Напишите функцию `std::vector<Time> getTimesFromString(const std::string& s)`, которая будет принимать строку в формате "`hh:mm:ss ... hh:mm:ss`", где за место букв должны стоять некоторые числа. Например, строка может иметь вид "`11:20:05 05:45:30 22:10:45`". Функция должна возвращать вектор времен, соответствующий временам в строке.
- Напишите функцию `Time sumTimes(const std::vector<Time>& v)`, которая будет суммировать все времена и возвращать эту сумму. Для суммирования времён используйте перегруженный оператор `+` класса `Time`.

Использовать функции можно следующим образом:

```
std::vector<Time> v = getTimesFromString("11:20:05 05:45:30 22:10:45");
v.push_back(Time("01:10:30"));
Time s = sumTimes(v);
std::cout << s << std::endl;
```

В результате исполнения этого участка кода на экран должно напечататься `16:26:50`.

## Задача 5. Максимум в векторе

Напишите шаблонную функцию, которая будет принимать на вход вектор элементов некоторого типа и возвращать максимальный элемент этого вектора. Протестируйте данную функцию на векторах, содержащих объекты следующих типов: `int`, `float`, `std::string` и `std::pair<int, int>`.

## Задача 6. Разбиение на пары

Напишите шаблонную функцию, которая будет принимать на вход контейнер (`vector`, `array`, `string`, `span` или `string_view`) и возвращать вектор пар нечётных и чётных элементов. Если в контейнере было нечётное количество элементов, то второй элемент последней пары должен быть инициализирован с помощью `value-инициализации`.

аргумент	возвращаемое значение
<code>std::vector{10, 20, 30, 40, 50}</code>	<code>{{10, 20}, {30, 40}, {50, 0}}</code>
<code>std::array&lt;std::string, 4&gt;{"cat", "dog", "mouse", "lion"}</code>	<code>{{"cat", "dog"}, {"mouse", "lion"}}</code>
<code>"Hello"</code>	<code>{{'H', 'e'}, {'l', 'l'}, {'o', '\0'}}</code>

Протестировать функцию можно в файле `code/test_pairing.cpp`.

## Задача 7. 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;
a = a * 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 8. Менеджер

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:

```
Manager<std::string> a;
a.allocate();

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

a.construct("Sapere Aude");
cout << a.get() << endl;
a.destruct();

a.deallocate();
```

## Задача 9. 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