# Шаблонные классы.
# Non type параметры.
# Специализация классов.

Мстоян Амиран

# Разбор домашнего задания

```
class ArrayInt
{
private:
    int m_length;
    int *m_data;

public:
    ArrayInt()
    {
        m_length = 0;
        m_data = nullptr;
    }

    ArrayInt(int length)
    {
        assert(length > 0);
        m_data = new int[length];
        m_length = length;
    }

    ~ArrayInt()
    {
        delete[] m_data;
    }

    void Erase()
    {
        delete[] m_data;
        // Присваиваем значение nullptr для m_data, чтобы на выходе не получить висячий указатель!
        m_data = nullptr;
        m_length = 0;
    }

    int& operator[](int index)
    {
        assert(index >= 0 && index < m_length);
        return m_data[index];
    }

    int getLength() { return m_length; }
};
```

```
class ArrayDouble
{
private:
    int m_length;
    double *m_data;

public:
    ArrayDouble()
    {
        m_length = 0;
        m_data = nullptr;
    }

    ArrayDouble(int length)
    {
        assert(length > 0);
        m_data = new double[length];
        m_length = length;
    }

    ~ArrayDouble()
    {
        delete[] m_data;
    }

    void Erase()
    {
        delete[] m_data;
        // Присваиваем значение nullptr для m_data, чтобы на выходе не получить висячий указатель!
        m_data = nullptr;
        m_length = 0;
    }

    double & operator[](int index)
    {
        assert(index >= 0 && index < m_length);
        return m_data[index];
    }

    int getLength() { return m_length; }
};
```

```cpp
template <class T> // это шаблон класса с T вместо фактического (передаваемого) типа данных
class Array
{
private:
    int m_length;
    T *m_data;

public:
    Array()
    {
        m_length = 0;
        m_data = nullptr;
    }

    Array(int length)
    {
        m_data = new T[length];
        m_length = length;
    }

    ~Array()
    {
        delete[] m_data;
    }

    void Erase()
    {
        delete[] m_data;
        // Присваиваем значение nullptr для m_data, чтобы на выходе не получить висячий указатель!
        m_data = nullptr;
        m_length = 0;
    }


    T& operator[](int index)
    {
        assert(index >= 0 && index < m_length);
        return m_data[index];
    }

    // Длина массива всегда является целочисленным значением, она не зависит от типа элементов массива
    int getLength(); // определяем метод и шаблон метода getLength() ниже
};
```

```
template <typename T> // метод, определенный вне тела класса, нуждается в собственном определении шаблона метода
int Array<T>::getLength() { return m_length; } // обратите внимание, имя класса - Array<T>, а не просто Array
```

# Non type параметр

**Параметр non-type в шаблоне** — это специальный параметр шаблона, который заменяется не типом данных, а конкретным значением. Этим значением может быть:

- целочисленное значение или **перечисление**;
- **указатель** или ссылка на объект класса;
- указатель или **ссылка** на функцию;
- указатель или ссылка на метод класса;
- **std::nullptr_t**.

```cpp
#include <iostream>

template <class T, int size> // size является параметром non-type в шаблоне класса
class StaticArray
{
private:
    // Параметр non-type в шаблоне класса отвечает за размер выделяемого массива
    T m_array[size];

public:
    T* getArray();

    T& operator[](int index)
    {
        return m_array[index];
    }
};

// Синтаксис определения шаблона метода и самого метода вне тела класса с параметром non-type
template <class T, int size>
T* StaticArray<T, size>::getArray()
{
    return m_array;
}
```

# Явная специализация шаблона класса

```
template <class T>
class Repository
{
private:
    T m_value;
public:
    Repository(T value)
    {
        m_value = value;
    }

    ~Repository()
    {
    }

    void print()
    {
        std::cout << m_value << '\n';
    }
};
```

```
int main()
{
    // Инициализируем объекты класса
    Repository<int> nValue(7);
    Repository<double> dValue(8.4);

    // Выводим значения объектов класса
    nValue.print();
    dValue.print();
}
```