

# Homework

## Part 1 (minimal vector<T> implementation)

You need to implement a custom vector (`vec<T>`) with minimal functionality. You have to use either `operator new/operator delete` or `malloc/free`. That means you have to work with uninitialized (raw) memory. Don't mix with initialized new, use only the uninitialized version and construct objects with `placement new`.

```
#define INITIAL_SIZE 10

template <typename T>
class vec
{
public:
    using value_type          = T;
    using size_type           = size_t;
    using reference           = value_type&;
    using const_reference     = const value_type&;

    // You have to implement (at least) all the following methods
    vec();
    explicit vec(size_type);
    vec(size_type sz, const value_type&);
    vec(const vec& v2);
    vec(vec&& v2) noexcept;
    vec(const std::vector<T>&);
    explicit operator std::vector<T>() const;
    ~vec() noexcept;
    vec& operator=(const vec& v2);
    vec& operator=(vec&& v2) noexcept;
    template <typename Q>
    friend void swap_(vec<Q>&, vec<Q>&);
    void resize(size_type);
    void reserve(size_type);
    size_type size() const;
    size_type capacity() const;
    void push_back(const value_type&);
    void pop_back();
    reference operator[](size_type);
    const_reference operator[](size_type) const;
    value_type* cbegin();
    value_type* cend();
private:
    value_type* data;
    size_t capacity_;
    size_t size_used;
```

```

        static const size_type default_init_capacity = INITIAL_SIZE;
};
template <typename T>
bool operator==(vec<T>& v1, vec<T>& v2)
{
    return v1.size() == v2.size() && v1.capacity() ==
v2.capacity() && equal(v1.cbegin(), v1.cend(), v2.cbegin(),
v2.cend());
}

```

Header file “vec\_tests.h” includes 10 test functions, if implemented correctly all the tests should print “true”.

## Part 1 (algorithmic problems on vec<T>)