

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
Национальный исследовательский университет ИТМО

МЕГАФАКУЛЬТЕТ ТРАНСЛЯЦИОННЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММИРОВАНИЯ

ЛАБОРАТОРНАЯ РАБОТА №4
По дисциплине «Программирование»
Лабораторная работа. Алгоритмы и итераторы

Выполнил Кудашев И.Э
(Фамилия Имя Отчество)

Проверил Повышев В.В
(Фамилия Имя Отчество)

Санкт-Петербург, 2021г

УСЛОВИЕ ЛАБОРАТОРНОЙ

Требуется реализовать следующие обобщенные алгоритмы.

1. **all_of** - возвращает true, если все элементы диапазона удовлетворяют некоторому предикату. Иначе false
2. **any_of** - возвращает true, если хотя бы один из элементов диапазона удовлетворяет некоторому предикату. Иначе false
3. **none_of** - возвращает true, если все элементы диапазона не удовлетворяют некоторому предикату. Иначе false
4. **one_of** - возвращает true, если ровно один элемент диапазона удовлетворяет некоторому предикату. Иначе false
5. **is_sorted** - возвращает true, если все элементы диапазона находятся в отсортированном порядке относительно некоторого критерия
6. **is_partitioned** - возвращает true, если в диапазоне есть элемент, делящий все элементы на удовлетворяющие и не удовлетворяющие некоторому предикату. Иначе false.
7. **find_not** - находит первый элемент, не равный заданному
8. **find_backward** - находит первый элемент, равный заданному, с конца
9. **is_palindrome** - возвращает true, если заданная последовательность является палиндромом относительно некоторого условия. Иначе false.

КОД

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

//LAB_ALGORITHMS
namespace CustomAlgorithms {
    template<typename T, typename P>
    bool all_of(const T &begin, const T &end, P &p) {
        for (T i = begin; i != end; i++) {
            if (!p(*i)) {
                return false;
            }
        }
        return true;
    };

    template<typename T, typename P>
    bool any_of(const T &begin, const T &end, const P &p) {
        for (T i = begin; i < end; i++) {
            if (p(*i)) {
                return true;
            }
        }
        return true;
    }

    template<typename T, typename P>
    bool none_of(const T &begin, const T &end, const P &p) {
        return !(CustomAlgorithms::all_of(begin, end, p));
    }

    template<typename T, typename P>
    bool one_of(const T &begin, const T &end, const P &p) {
        int count_of_agrees = 0;
        for (T i = begin; i < end; i++) {
            if (p(*i)) {
                count_of_agrees++;
            }
        }

        if (count_of_agrees == 1) {
```

```

        return true;
    } else {
        return false;
    }
}

template<typename T, typename P>
bool is_sorted(const T &begin, const T &end, const P &p) {
    for (T i = begin; i < end; i++) {
        if (p(*i, *(i + 1))) {
            return false;
        }
    }
    return true;
}

template<typename T, typename P>
bool is_partitioned(const T &begin, const T &end, const P p) {
    T start = begin;
    T finish = end - 1;
    bool isCorrect = false;
    for (T i = begin; i < end; ++i) {
        if (!isCorrect) {
            if (p(*start++, *finish))
                isCorrect = true;
        } else if (start == finish) {
            break;
        } else if (p(*start++) != p(*start))
            return false;
    }
    return isCorrect;
}

template<typename T1, typename T2>
T2 find_not(const T1 &begin, const T1 &end, T2 value) {
    for (T1 i = begin; i < end; i++) {
        if (*i != value) {
            return *i;
        }
    }
    return value;
}

template<typename T1, typename T2>
T2 find_backward(const T1 &begin, const T1 &end, const T2 value) {
    for (T1 i = begin; i < end; i++) {
        if (*i == value) {
            return *i;
        }
    }
}

```

```

        return value;
    }

    template<typename T, typename Predicate>
    bool is_palindrome(const T &begin, const T &end, Predicate foo) {
        bool isCorrect = false;
        T PBegin = begin, PEnd = end - 1;
        while (PBegin < PEnd) {
            if (foo(*PBegin++, *PEnd--)) {
                isCorrect = true;
            } else return false;
        }
        return isCorrect;
    }
}

// TEST CLASS
template<typename T>
class Nums {
private:
    T a;
    T b;

public:
    Nums() = default;

    Nums(T a, T b) {
        this->a = a;
        this->b = b;
    }

    T get_a() const {
        return a;
    }

    T get_b() const {
        return b;
    }
};

//PREDICTS
template<class T>
bool isNull(T t) {
    return (t == 0);
};

template<class T>

```

```

bool is_even(T t) {
    return t % 2 == 0;
}

template<class T>
bool is_more(const T x, const T y) {
    return (x > y);
}

template<class T>
bool is_symmertic( T x, T y) {
    return (x == y);
}

template<class T, class P>
bool is_multiplyate_null(const T a, const P b) {
    auto result = a * b;
    if (result == 0) {
        return true;
    } else {
        return false;
    }
}

bool is_multiplyate_null_for_class(const Nums<int> num) {
    auto result = num.get_a() * num.get_b();
    if (result == 0) {
        return true;
    } else {
        return false;
    }
}

int main() {
    //DATA
    vector <Nums<int> VNums = {
        Nums<int>(0, 0),
        Nums<int>(0, 0),
        Nums<int>(0, 0),
        Nums<int>(0, 0),
    };

    vector <Nums<int> VNums2 = {
        Nums<int>(1, 1),
        Nums<int>(0, 0),
        Nums<int>(1, 1),
    };
}

```

```

    Nums<int>(1, 1),
};

vector<float> VFloat = {
    1.1,
    2.2,
    1.1
};

vector<int> VSorted = {
    1, 2, 3, 4, 5, 6, 7
};

vector<int> VNSorted = {
    4, 1, 0, -20, -500
};

vector<int> VEven = {
    10, 121, 123123, 454556
};


string str = 'abcba';

cout << '---RESULTS---';
cout << endl;
cout << 'all_of' << ' ' << CustomAlgorithms::all_of(VNums.cbegin(), VNums.cend(), is_multiply_null_for_class);
cout << endl;
cout << 'any_of' << ' ' << CustomAlgorithms::any_of(VNums.cbegin(), VNums.cend(), is_multiply_null_for_class);
cout << endl;
cout << 'none_of' << ' ' << CustomAlgorithms::none_of(VFloat.cbegin(), VFloat.cend(), isNull<float>);
cout << endl;
cout << 'one_of' << ' ' << CustomAlgorithms::one_of(VNums2.cbegin(), VNums2.cend(),
is_multiply_null_for_class);
cout << endl;
cout << 'is_sorted' << ' ' << CustomAlgorithms::is_sorted(VNSorted.cbegin(), VNSorted.cend(), is_more<int>);
cout << endl;
cout << 'is_partitioned' << ' ' << CustomAlgorithms::is_partitioned(VEven.cbegin(), VEven.cend(), is_even<int>);
cout << endl;
cout << 'find_not' << ' ' << CustomAlgorithms::find_not(VEven.cbegin(), VEven.cend(), int(10));
cout << endl;
cout << 'find_backward' << ' ' << CustomAlgorithms::find_backward(VSorted.crbegin(), VSorted.crend(), int(7));
cout << endl;
cout << 'is_palindrome' << ' ' << CustomAlgorithms::is_palindrome(VFloat.begin(), VFloat.end(),
is_symmetric<float> );

```

```
return 0;  
}
```

Кудашев И.Э

A small, square, light-colored image containing a handwritten signature in dark ink. The signature is stylized and appears to be the initials or full name of the person mentioned in the adjacent text block.