Министр науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет информационных технологий и программирования

Лабораторная работа № 4

STL. Алгоритмы. Итераторы.

Выполнила студентка группы № M3106
Шеин Максим Андреевич
Подпись:

Проверил:
Повышев Владислав Вячеславович

Санкт-Петербург

2020

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.

Каждый алгоритм должен быть выполнен в виде шаблонной функции, позволяющей взаимодействовать со стандартными контейнерами STL с помощью итераторов. Предикаты, условия, операторы сравнения должны быть параметризованы.
При сдаче работы требуется продемонстрировать работу алгоритмов как на стандартных, так и на пользовательских типах данных, например CPoint, CRational, далее работает ваша индивидуальная (не “коллективная”) фантазия.

Решение

```
#include <iostream>
#include <iterator>
#include <vector>
#include <list>
#include <math.h>

template<class T>
bool Hmmmm(T x)
{
   return (x * 2 == 2 * x);
}

template<class T>
bool Sort(T x1, T x2)
{
   return (x1 <= x2);
}

//1) Возвращает true, если все элементы диапазона удовлетворяют некоторому предикату. Иначе false
template <typename T, typename Function>
bool all_of(const T& begin, const T& end, Function function)
{
```

```
    for (T iterator = begin; iterator != end; iterator++)
    {
        if (!function(*iterator))
        {
            return false;
        }
    }
    return true;
}

//2) Возвращает true, если хотя бы один из элементов диапазона удовлетворяет некоторому предикату. Иначе false
template <typename T, typename Function>
bool any_of(const T& begin, const T& end, Function function)
{
    for (T iterator = begin; iterator != end; iterator++)
    {
        if (function(*iterator))
        {
            return true;
        }
    }
    return false;
}

//3) Возвращает true, если все элементы диапазона не удовлетворяют некоторому предикату. Иначе false
template <typename T, typename Function>
bool none_of(const T& begin, const T& end, Function function)
{
    for (T iterator = begin; iterator != end; iterator++)
    {
        if (function(*iterator))
        {
            return false;
        }
    }
    return true;
}

//4) Возвращает true, если ровно один элемент диапазона удовлетворяет некоторому предикату. Иначе false
template <typename T, typename Function>
bool one_of(const T& begin, const T& end, Function function)
{
    int count = 0;
    for (T iterator = begin; iterator != end; iterator++)
    {
        if (function(*iterator))
        {
            count++;
        }
```

```
    }
    return count == 1;
}

//5) Возвращает true, если все элементы диапазона находятся в отсортированном порядке относительно некоторого критерия
template <typename T, typename Function>
bool is_sorted(const T& begin, const T& end, Function function)
{
    for (T iterator = begin + 1; iterator != end; iterator++)
    {
        if (!function(*iterator, *(iterator - 1)))
        {
            return false;
        }
    }
    return true;
}
//6)возвращает true, если в диапазоне есть элемент, делящий все элементы на удовлетворяющие и не удовлетворяющие некоторому предикату. Иначе false.

template<typename T, typename Function> //Вспомогательный алгоритм.
constexpr T find_if_not(T begin, T end, Function function)
{
    for (; begin != end; ++begin)
    {
        if (!function(*begin))
        {
            return begin;
        }
    }
    return end;
}

//7) Находит первый элемент, не равный заданному
template <typename T, typename Element>
Element find_not(const T& begin, const T& end, Element element)
{
    for (T iterator = begin; iterator != end; iterator++)
    {
        if (*iterator != element)
        {
            return *iterator;
        }
    }
    return *end; // Согласно канонам обобщённых алгоритмов
}

//8) Находит первый элемент, равный заданному, с конца
template <typename T, typename Element>
Element find_backward(const T& begin, const T& end, Element element)
{
```

```
    for (T iterator = end - 1; iterator >= begin; iterator--)
    {
        if (*iterator == element)
        {
            return *iterator;
        }
    }
    return *end; // Согласно канонам обобщённых алгоритмов
}

//9) Возвращает true, если заданная последовательность является палиндромом относительно некоторого условия. Иначе false.
template <typename T, typename Function>
bool is_palindrome(const T& begin, const T& end, Function function)
{
    T start, finish;
    for (start = begin, finish = end - 1; start != end && finish != begin && function(*start) == function(*finish); start++, finish--)
    {

    }
    return start == end - 1 && finish == begin;
}

template <typename T>
void printArray(const T& begin, const T& end)
{
    for (T iter = begin; iter != end; iter++)
    {
        std::cout << *iter << " ";
    }
    std::cout << std::endl;
}

int main()
{
    setlocale(LC_ALL, "Russian");
    int x;

    std::vector<int> v = {2, 1, -3, 1, 2};
    std::cout << "Заданные значения: ";
    printArray(v.begin(), v.end());

    std::cout << "1)ALL_OF = " << (all_of(v.begin(), v.end(), Hmmmm<int>) ? "TRUE" : "FALSE") << std::endl;

    std::cout << "2)ANY_OF = " << (any_of(v.begin(), v.end(), Hmmmm<int>) ? "TRUE" : "FALSE") << std::endl;

    std::cout << "3)NONE_OF = " << (none_of(v.begin(), v.end(), Hmmmm<int>) ? "TRUE" : "FALSE") << std::endl;
```

```
    std::cout << "4)ONE_OF = " << (one_of(v.begin(), v.end(), Hmmmm<int>) ? "TRUE" :
"FALSE") << std::endl;

    std::cout << "5)IS_SORTED = " << (is_sorted(v.begin(), v.end(), Sort<int>) ? "TRUE" :
"FALSE") << std::endl;

    std::cout << "\n7)Задайте элемент для find_not: ";
    std::cin >> x;
    std::cout << "FIND_NOT("<< x <<") = " << find_not(v.begin(), v.end(), x) << std::endl;

    std::cout << "\n8)Задайте элемент для find_backward: ";
    std::cin >> x;
    std::cout << "FIND_BACKWARD(" << x <<") = " << find_backward(v.begin(), v.end(), x)
<< std::endl;

    std::cout << "\n9)IS_PALINDROME = " << (is_palindrome(v.begin(), v.end(),
Hmmmm<int>) ? "TRUE" : "FALSE") << std::endl;
    return 0;
}
```