

**Федеральное государственное автономное
образовательное учреждение высшего
образования**

**«Национальный исследовательский
университет ИТМО»**

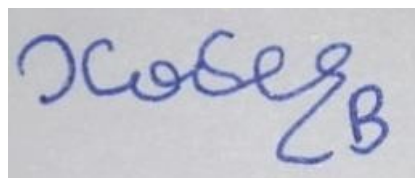
**Факультет информационных технологий
и Программирования**

Программирование

Лабораторная работа № 3

Выполнил студент группы № М3104
Хобер Владислав Алимович

Подпись:



Санкт-Петербург

2022

Лабораторная №3

Реализовать кольцевой буфер в виде stl-совместимого контейнера (например, может быть использован с стандартными алгоритмами), обеспеченного итератором произвольного доступа. Реализация не должна использовать ни один из контейнеров STL. Буфер должен обладать следующими возможностями:

1. Вставка и удаление в конец
2. Вставка и удаление в начало
3. Доступ в конец, начало
4. Доступ по индексу
5. Изменение емкости
6. Алгоритмы

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`.

main.cpp

```
#include <iostream>
#include<vector>
#include<algorithm>
#include "circular_buffer.h"
#include "algorithms.h"
#include "point.h"

template<typename T>
```

```

bool function(T a)
{
    return a == 50;
}

void pow(int &i)
{ i *= i; }

template<typename T>
bool sorted(T a, T b)
{
    if (a > b)
        return true;
    return false;
}

template<typename T>
bool zero(T a)
{
    if (a == 0)
        return true;
    return false;
}

template<typename T>
bool greater(T a)
{
    if (a < 20)
        return true;
    return false;
}

template<typename T>
bool func(T a, T b)
{
    if (a == b)
        return true;
    return false;
}

int main()
{
    circular_buffer<int> buffer(8);
    std::cout << "Capacity of the buffer - " << buffer.capacity() <<
std::endl;
    for (int i = 0; i < 4; i++)
        buffer.push_back(i);
    for (int i = 4; i < 8; i++)
        buffer.push_front(i);
    std::cout << buffer.front() << " - front element" << std::endl;
    std::cout << buffer.back() << " - back element" << std::endl;
    buffer.print();
    buffer.erase_front();
    buffer.erase_back();
}

```

```

std::cout << "Gonna delete some elements" << std::endl;
buffer.print();
std::cout << "Create a new buffer" << std::endl;
circular_buffer<int> new_buffer(buffer);
new_buffer.print();
for (int i = 8; i < 12; i++)
{
    new_buffer.push_front(i);
}
for (int i = 12; i < 16; i++)
{
    new_buffer.push_back(i);
}
std::cout << "Try to add new elements" << std::endl;
new_buffer.print();
std::cout << "Resize a buffer" << std::endl;
new_buffer.resize(3);
new_buffer.print();
new_buffer.resize(10);
for (int i = 16; i < 26; i++)
    new_buffer.push_back(i);
new_buffer.print();
std::cout << "Using an iterator with STL algorithms" << std::endl;
circular_buffer<int>::iterator it = std::find(new_buffer.begin(),
new_buffer.end(), 25);
std::cout << *it << std::endl;
std::cout << (bool) std::any_of(new_buffer.begin(), new_buffer.end(),
function<int>) << std::endl;
std::for_each(new_buffer.begin(), new_buffer.end(), pow);
new_buffer.print();
circular_buffer<int>::iterator it2 = new_buffer.begin();
std::cout << it2[2] << std::endl;
std::cout << "Using my algorithms with different types" << std::endl;
circular_buffer<int>::iterator it3 = Find_backward(new_buffer.begin(),
new_buffer.end(), 400);
std::cout << *it3 << std::endl;

std::vector<int> vec;
vec.push_back(0);
vec.push_back(0);
vec.push_back(0);
vec.push_back(0);
vec.push_back(1);
auto find_it = Find_not(vec.begin(), vec.end(), 0);
std::cout << *find_it << std::endl;
std::cout << is_Sorted(vec.begin(), vec.end(), sorted<int>) <<
std::endl;
std::cout << is_Partioned(vec.begin(), vec.end(), zero<int>) <<
std::endl;
Point point[15];
for (int i = 0; i < 15; i++)
{
    point[i] = Point(i, i + 1);
}

```

```

    std::cout << All_of(point, point + 15, greater<Point>) << std::endl;
    Point points[4];
    points[0] = Point(4, 5);
    points[3] = Point(4, 5);
    points[1] = Point(2, 3);
    points[2] = Point(2, 3);
    std::cout << is_Palindrome(point, point + 15, func<Point>) <<
std::endl;
    std::cout << is_Palindrome(points, points + 4, func<Point>) <<
std::endl;

    return 0;
}

```

algorithms.h

```

#ifndef LAB3_ALGORITHMS_H
#define LAB3_ALGORITHMS_H

template<class iterator, class predicate>
bool All_of(iterator first, iterator last, predicate pred)
{
    while (first != last)
    {
        if (!pred(*first))
            return false;
        ++first;
    }
    return true;
}

template<class iterator, class predicate>
bool Any_of(iterator first, iterator last, predicate pred)
{
    while (first != last)
    {
        if (pred(*first))
            return true;
        ++first;
    }
    return false;
}

template<class iterator, class predicate>
bool None_of(iterator first, iterator last, predicate pred)
{
    while (first != last)
    {
        if (pred(*first))

```

```

        return false;
        ++first;
    }
    return true;
}

template<class iterator, class predicate>
bool One_of(iterator first, iterator last, predicate pred)
{
    int k = 0;
    while (first != last)
    {
        if (pred(*first))
        {
            k++;
        }
        ++first;
    }
    if (k == 1)
        return true;
    return false;
}

template<class iterator, class sort_cmp>
bool is_Sorted(iterator first, iterator last, sort_cmp cmp)
{
    if (first == last)
        return true;
    iterator next = first;
    while (++next != last)
    {
        if (!cmp(*first, *next))
        {
            return false;
        }
        ++first;
    }
    return true;
}

template<class iterator, class predicate>
bool is_Partioned(iterator first, iterator last, predicate pred)
{
    while (first != last && pred(*first))
    {
        ++first;
    }
    while (first != last)
    {
        if (pred(*first))
            return false;
        ++first;
    }
    return true;
}

```

```

}

template<class iterator, typename T>
iterator Find_not(iterator first, iterator last, T value)
{
    while (first != last)
    {
        if (*first != value)
            return first;
        ++first;
    }
    return last;
}

template<class iterator, typename T>
iterator Find_backward(iterator first, iterator last, T value)
{
    iterator next = last;
    --next;
    while (next != first)
    {
        if (*next == value)
            return next;
        --next;
    }
    if (*next == value)
        return next;
    return last;
}

template<class iterator, class predicate>
bool is_Palindrome(iterator first, iterator last, predicate pred)
{
    iterator next = last;
    --next;
    while (next != first && next > first)
    {
        if (!pred(*first, *next))
            return false;
        ++first;
        --next;
    }
    return true;
}

#endif

```

circular_buffer.h

```

#ifndef LAB3_CIRCULAR_BUFFER_H
#define LAB3_CIRCULAR_BUFFER_H

template<typename T>
class circular_buffer
{
public:
    class iterator : public std::iterator<std::random_access_iterator_tag,
T>
    {
        friend class circular_buffer;

    public:
        iterator()
        {}

        iterator(T *it_)
            : it(it_)
        {}

        T &operator*()
        {
            return *it;
        }

        int operator-(iterator other)
        {
            return it - other.it;
        }

        iterator operator+(const int n)
        {
            it = it + n;
            return *this;
        }

        iterator operator-(const int n)
        {
            it = it - n;
            return *this;
        }

        iterator operator+=(const int n)
        {
            it = it + n;
            return *this;
        }

        iterator operator-=(const int n)
        {
            it = it - n;
            return *this;
        }
    }
};

```



```

}

iterator operator++()
{
    ++it;
    return *this;
}

iterator operator--()
{
    --it;
    return *this;
}

bool operator!=(circular_buffer<T>::iterator other)
{
    return it != other.it;
}

bool operator==(circular_buffer<T>::iterator other)
{
    return it == other.it;
}

bool operator<(circular_buffer<T>::iterator other)
{
    return it < other.it;
}

bool operator>(circular_buffer<T>::iterator other)
{
    return it > other.it;
}

bool operator<=(circular_buffer<T>::iterator other)
{
    return it <= other.it;
}

bool operator>=(circular_buffer<T>::iterator other)
{
    return it >= other.it;
}

T operator[](const int n)
{
    return *(it + n);
}

friend iterator operator+(const int n, iterator &it_)
{
    return iterator(it_ + n);
}

```

```

private:
    T *it;
};

circular_buffer()
{}

circular_buffer(size_t n)
{
    Begin = (T *) malloc(sizeof(T) * (n + 1));
    End = Begin;
    Border = Begin + n;
    Capacity = n;
}

circular_buffer(const circular_buffer &other)
{
    Begin = (T *) malloc(sizeof(T) * (other.capacity() + 1));
    T *pointer = Begin;
    End = Begin;
    for (size_t i = 0; i < other.size(); i++)
    {
        *pointer = other[i];
        ++pointer;
        ++End;
    }
    Capacity = other.Capacity;
    Size = other.size();
    Border = Begin + Capacity;
}

int size() const
{
    return Size;
}

int capacity() const
{
    return Capacity;
}

void resize(size_t n)
{
    T *new_begin = (T *) malloc(sizeof(T) * (n + 1));
    T *new_end = new_begin;
    T *pointer = new_begin;
    size_t k = 0;
    for (size_t i = 0; i < this->size(); i++)
    {
        if (k == n)
            break;
        *pointer = (*this)[i];
        ++pointer;
        ++new_end;
    }
}

```

```

        k++;
    }
    free(Begin);
    Begin = new_begin;
    End = new_end;
    Capacity = n;
    Size = k;
    Border = Begin + n;
}

void push_back(T value)
{
    if (End == Border)
    {
        erase_front();
    }
    *End = value;
    End++;
    Size++;
}

void push_front(T value)
{
    if (Size == 0)
    {
        this->push_back(value);
        return;
    }
    T *pointer = End;
    while (pointer != Begin)
    {
        T *value = pointer;
        --value;
        *pointer = *value;
        --pointer;
    }
    *pointer = value;
    if (End != Border)
    {
        End++;
        Size++;
    }
}

void erase_back()
{
    if (Begin != End)
    {
        --End;
        Size--;
    }
}

```

```

void erase_front()
{
    if (Begin == End)
        return;
    T *pointer = Begin;
    T *end = End;
    --end;
    while (pointer != end)
    {
        T *value = pointer;
        ++value;
        *pointer = *value;
        ++pointer;
    }
    End--;
    Size--;
}

T front() const
{
    return *Begin;
}

T back() const
{
    T *pointer = End;
    --pointer;
    return *pointer;
}

void print() const
{
    for (size_t i = 0; i < this->size(); i++)
    {
        std::cout << (*this)[i] << ' ';
    }
    std::cout << std::endl;
}

~circular_buffer()
{
    free(Begin);
}

circular_buffer &operator=(const circular_buffer &buffer) = delete;

T operator[](size_t n) const
{
    T *pointer = Begin;
    pointer = pointer + n;
    return *pointer;
}

```

```

    iterator begin() const
    {
        return iterator(Begin);
    }

    iterator end() const
    {
        return iterator(End);
    }

private:
    size_t Size = 0;
    size_t Capacity = 0;
    T *Begin;
    T *End;
    T *Border;
};

#endif

```

point.h

```

#ifndef LAB3_POINT_H
#define LAB3_POINT_H

#ifndef LAB1_POINT_H
#define LAB1_POINT_H

class Point
{
public:
    Point(int x = 0, int y = 0)
        : x_(x), y_(y)
    {}

    Point(const Point &other)
        : x_(other.x_), y_(other.y_)
    {}

    ~Point()
    {}

    Point &operator=(const Point &other)
    {
        if (&other == this)
            return *this;
        x_ = other.x_;
        y_ = other.y_;
        return *this;
    }
}

```

```

bool operator!=(const Point &other)
{
    if (this->x_ == other.x() && this->y_ == other.y())
        return false;
    else
        return true;
}

bool operator==(const Point &other)
{
    if (x_ == other.x() && y_ == other.y())
        return true;
    return false;
}

bool operator>(int n)
{
    if (x_ > n && y_ > n)
        return true;
    return false;
}

bool operator<(int n)
{
    if (x_ < n && y_ < n)
        return true;
    return false;
}

void x_change(int x)
{
    this->x_ = x;
}

void y_change(int y)
{
    this->y_ = y;
}

int x() const
{
    return x_;
}

int y() const
{
    return y_;
}

private:
    int x_;
    int y_;
};

```

```
#endif  
#endif
```

Ввод и вывод

Программа показывает результат взаимодействия с кольцевым буфером.

Вывод

Используя принципы ООП, я выполнил предложенное мне задание, научился создавать совместимые с STL классы.