Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет информационных технологий и программирования

Лабораторная работа № 7

Кольцевой буфер

Выполнил студент группы № M3111
Гаврилов Алексей Евгеньевич
Подпись:

Проверил:
Повышев Владислав Вячеславович

Санкт-Петербург
2022

Текст задания

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

Буфер должен обладать следующими возможностями:

1. Вставка и удаление в конец
2. Вставка и удаление в начало
3. Вставка и удаление в произвольное место по итератору
4. Доступ в конец, начало
5. Доступ по индексу
6. Изменение капасити

Решение с комментариями

//main

```cpp
#include <iostream>
#include "Circle_Buffer.h"
#include <algorithm>
using namespace std;
int main() {
    Circle_Buffer<int> buff(6);
    cout << "1 2 3 4 5 6 7 -> 7 2 3 4 5 6" << endl;
    cout << endl;
    for (int i = 1; i < 8; i++) {
        buff.push_front(i);
        for (int i = 0; i < 6; i++)
            cout << buff[i] << " ";
        cout << endl;
    }
    cout << endl;
    cout << "1 2 3 4 5 6 7 -> 7 6 5 4 3 2" << endl;
    cout << endl;
    for (int i = 1; i < 8; i++) {
        buff.push_back(i);
        for (int i = 0; i < 6; i++)
            cout << buff[i] << " ";
        cout << endl;
    }
    cout << endl;
    cout << "7 6 5 4 3 2 -> 0 0 0 0 0 0" << endl;
    buff.clean_buffer();
    cout << endl;
    for (int i = 0; i < 6; i++)
        cout << buff[i] << " ";
    cout << endl << endl;

    for (int i = 0; i < 6; i++) {
        buff.push_front(i + 1);
        cout << buff[i] << " ";
    }
    cout << endl << endl;
    cout << "First and last elements: 1 6" << endl << endl;
    cout << buff.front() << " ";
    cout << endl;
    cout << buff.back() << " ";
    cout << endl << endl;
    cout << "Change copacity on 4. Result -> 1 2 3 4 1 1" << endl << endl;
    buff.change_copacity(4);
    for (int i = 0; i < 6; i++) {
        cout << buff[i] << " ";
    }
    cout << endl << endl;
    cout << "Pop front and back elements -> 0 2 3 0 0 0" << endl << endl;
    buff.pop_back();
    buff.pop_front();
    for (int i = 0; i < 6; i++) {
        cout << buff[i] << " ";
    }
    cout << endl;
    //sort(buff[0], buff[1]);
    cout << "0 2 3 0 0 0 -> 0 0 0 0 0 0" << endl;
    buff.clean_buffer();
    cout << endl;
    for (int i = 0; i < 6; i++)
        cout << buff[i] << " ";
    cout << endl << endl;
```

```
    for (int i = 4; i > 0; --i) {
        buff.push_front(i);
    }
    for (int i = 0; i < 4; i++)
        cout << buff[i] << " ";
    sort(buff.begin(), buff.end());
    cout << endl;
    for (int i = 0; i < 4; i++)
        cout << buff[i] << " ";

    return 0;

}
```

//fun.h

```
#ifndef CURCLE_BUFFER_CIRCLE_BUFFER_H
#define CURCLE_BUFFER_CIRCLE_BUFFER_H
template<class T>
class Circle_Buffer {
private:
    int copacity = 0;
    int head = 0;
    int tail = 0;
    T *buff;
public:
    Circle_Buffer(int size) {
        copacity = size;
        tail = 0;
        head = 0;
        buff = new T[copacity];
        for (int i = 0; i < copacity; i++)
            buff[i] = 0;
    }

    ~Circle_Buffer() {
        delete[] buff;
    }
    class Iterator : public std::iterator<std::random_access_iterator_tag,
T>{
        T *p;
    public:

        Iterator()=default;

        explicit Iterator(T *temp) {
            p = temp;
        }
        Iterator(const Iterator &iterator): p(iterator.p){}

        ~Iterator() = default;

        Iterator operator+(int n){
            Iterator tmp = *this;
            tmp += n;
            return tmp;
        }

        Iterator &operator+=(int n){
            p += n;
            return *this;
        }
```

```
        Iterator operator-(int n){
            Iterator tmp = *this;
            tmp -= n;
            return tmp;
        }

        Iterator &operator-=(int n){
            p -=n;
            return *this;
        }

        T operator-(Iterator &other){
            return this->p - other.p;
        }

        T operator+(Iterator &other){
            return this->p + other.p;
        }

         Iterator &operator++(){
                p++;
                return *this;
        }

         Iterator &operator++(int){
            Iterator tmp = *this;
            p++;
            return  tmp;
        }

         Iterator &operator--(int){
            Iterator tmp = *this;
            p--;
            return tmp;
        }

         Iterator &operator--(){
            p--;
            return *this;
        }

        bool operator==(const Iterator &other){
            return p == other.p;
        }

        bool operator!=(const Iterator &other){
            return p != other.p;
        }

        T &operator *() const{
            return *p;
        }

        T *operator->(){
            return p;
        }

        bool operator<(Iterator other){
            return this->p < other.p;
        }

        bool operator>(Iterator other){
            return this->p > other.p;
        }
```

```
        bool operator<=(Iterator other){
            return this->p <= other.p;
        }

        bool operator>=(Iterator other){
            return this->p >= other.p;
        }
    };
    Iterator begin(){
        return Iterator(buff);
    }

    Iterator end(){
        T *end = (copacity + buff);
        return Iterator(end);
    }

    int length() {
        return copacity;
    }

    T &operator[](int i) {
        return i < copacity ? buff[i] : buff[0];
    }

    T &operator[](Iterator i) {
        return *i;
    }

    void push_front(const T &value) {
        buff[head] = value;
        head = head + 1 >= copacity ? 0 : head + 1;
    }

    void push_back(const T &value) {
        buff[tail] = value;
        tail = tail - 1 < 0 ? copacity - 1 : tail - 1;
    }

    T back() {
        return buff[copacity - 1];
    }

    T front() {
        return buff[0];
    }

    bool pop_back() {
        if (copacity > 0) {
            T temp = buff[copacity - 1];
            buff[copacity - 1] = 0;
            head = head - 1 < 0 ? copacity - 1 : head - 1;
            return true;
        } else
            return false;
    }

    bool pop_front() {
        if (copacity > 0) {
            T temp = buff[0];
            buff[0] = 0;
            tail = tail - 1 < 0 ? 0 : tail - 1;
            return true;
```

```
        } else
            return false;
    }

    void change_copacity(int size) {
        T *new_buff = new T[size];
        for (int i = 0; i < size; i++)
            new_buff[i] = i < copacity ? buff[i] : 0;
        buff = new_buff;
        copacity = size;
    }

    void clean_buffer() {
        head = 0;
        tail = 0;
        for (int i = 0; i < copacity; i++)
            buff[i] = 0;
    }
};
#endif
```