Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Информационных технологий и программирования

Программирование на C++

Работа: Лабораторная работа №7. Кольцевой буфер.

Выполнил: Гаджиев Саид M3115

Санкт-Петербург

2023

Лабораторная работа №7

Реализовать кольцевой буфер в виде stl-совместимого контейнера (например, может быть использован со 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. Вставка и удаление в произвольное место по итератору

4. Доступ в конец, начало

5. Доступ по индексу

6. Изменение капасити

Решение:

File “buffer.h”:

```
//
// Created by Redmibook on 4/23/2023.
//
#ifndef BUFFER_H
#define BUFFER_H

#include <iostream>

template<typename T>
class Buffer {
    private:
        T *DATA;
        int CAPACITY;
        int SIZE;
        int HEAD;
        int TAIL;

    public:
        class Iterator {
            private:
                T *ptr;

            public:
                Iterator(T *p = nullptr) {
                    ptr = p;
                }

                T &operator*() const {
                    return *ptr;
```

```
        }

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

        Iterator operator++(int) {
            Iterator temp = *this;
            ++ptr;
            return temp;
        }

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

        Iterator operator--(int) {
            Iterator temp = *this;
            --ptr;
            return temp;
        }

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

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

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

        Iterator operator-(int n) const {
            Iterator temp = *this;
            return temp -= n;
        }

        int operator-(const Iterator &rhs) const {
            return ptr - rhs.ptr;
        }

        bool operator==(const Iterator &rhs) const {
            return ptr == rhs.ptr;
        }

        bool operator!=(const Iterator &rhs) const {
            return !(*this == rhs);
        }
    };

    Buffer(int capacity) {
        CAPACITY = capacity;
        SIZE = 0;
        HEAD = 0;
        TAIL = 0;
        DATA = new T[capacity]();
    }
```

```
    Buffer(const Buffer &other) {
        CAPACITY = other.CAPACITY;
        SIZE = other.SIZE;
        HEAD = other.HEAD;
        TAIL = other.TAIL;
        DATA = new T[CAPACITY];
        for (int i = 0; i < SIZE; i++) {
            DATA[i] = other.DATA[(other.HEAD + i) % other.CAPACITY];
        }
    }

    Buffer &operator=(const Buffer &other) {
        if (this != &other) {
            delete[] DATA;
            CAPACITY = other.CAPACITY;
            HEAD = other.HEAD;
            TAIL = other.TAIL;
            SIZE = other.SIZE;
            DATA = new T[CAPACITY];
            for (int i = 0; i < SIZE; i++) {
                DATA[i] = other.DATA[(other.HEAD + i) % other.CAPACITY];
            }
        }
        return *this;
    }

    ~Buffer() {
        delete[] DATA;
    }

    int size() const {
        return SIZE;
    }

    bool empty() const {
        return SIZE == 0;
    }

    bool full() const {
        return SIZE == CAPACITY;
    }

    void push_back(const T &value) {
        if (full()) {
            throw std::out_of_range("Ring buffer is full. I CAN'T!!!");
        }
        DATA[TAIL] = value;
        TAIL = (TAIL + 1) % CAPACITY;
        SIZE++;
    }

    void push_front(const T &value) {
        if (full()) {
            throw std::out_of_range("Ring buffer is full. I CAN'T!!!");
        }
        HEAD = (HEAD - 1 + CAPACITY) % CAPACITY;
        DATA[HEAD] = value;
        SIZE++;
    }

    void pop_back() {
        if (empty()) {
            throw std::out_of_range("Ring buffer is empty. I CAN'T!!!");
```

```
            }
            if (TAIL == 0) {
                TAIL = CAPACITY - 1;
            } else {
                TAIL--;
            }
            SIZE--;
        }

        void pop_front() {
            if (empty()) {
                throw std::out_of_range("Ring buffer is empty. I CAN'T!!!");
            }
            HEAD++;
            if (HEAD == CAPACITY) {
                HEAD = 0;
            }
            SIZE--;
        }

        T &at(int index) const {
            if (index < 0 or index >= SIZE) {
                throw std::out_of_range("Index is out of range. I CAN'T!!!");
            }
            int actual_index = (HEAD + index) % CAPACITY;
            return DATA[actual_index];
        }

        T &operator[](int index) {
            return at(index);
        }

        Iterator begin() {
            return Iterator(&DATA[HEAD]);
        }

        Iterator end() {
            return Iterator(&DATA[TAIL]);
        }

        void insert(Iterator it, const T &value) {
            if (full()) {
                throw std::out_of_range("Ring buffer is full. I CAN'T!!!");
            }
            int index = it - begin();
            if (index < 0 or index > SIZE) {
                throw std::out_of_range("Invalid iterator. I CAN'T!!!");
            }
            if (index == 0) {
                push_front(value);
            } else if (index == SIZE) {
                push_back(value);
            } else {
                if (TAIL < HEAD) {
                    for (int i = SIZE; i > index; i--) {
                        DATA[(HEAD + i) % CAPACITY] = DATA[(HEAD + i - 1) %
CAPACITY];
                    }
                    DATA[(HEAD + index) % CAPACITY] = value;
                    HEAD = (HEAD - 1 + CAPACITY) % CAPACITY;
                } else {
                    for (int i = SIZE - 1; i >= index; i--) {
                        DATA[(TAIL - i - 1 + CAPACITY) % CAPACITY] =
DATA[(TAIL - i + CAPACITY) % CAPACITY];
```

```
                    }
                    DATA[(TAIL - index + CAPACITY) % CAPACITY] = value;
                    TAIL = (TAIL + 1) % CAPACITY;
                }
                SIZE++;
            }
        }

        void remove(Iterator it) {
            int index = it - begin();
            if (index < 0 or index >= SIZE) {
                throw std::out_of_range("Invalid iterator. I CAN'T!!!");
            }
            if (index == 0) {
                pop_front();
            } else if (index == SIZE - 1) {
                pop_back();
            } else {
                if (TAIL < HEAD) {
                    for (int i = index; i < SIZE - 1; i++) {
                        DATA[(HEAD + i) % CAPACITY] = DATA[(HEAD + i + 1) %
CAPACITY];
                    }
                    TAIL = (TAIL + 1) % CAPACITY;
                } else {
                    for (int i = index; i < SIZE - 1; i++) {
                        DATA[(TAIL - i + CAPACITY) % CAPACITY] = DATA[(TAIL -
i - 1 + CAPACITY) % CAPACITY];
                    }
                    HEAD = (HEAD + 1) % CAPACITY;
                }
                SIZE--;
            }
        }

        void resize(int new_capacity) {
            T *new_data = new T[new_capacity];
            int new_size = std::min(new_capacity, SIZE);
            for (int i = 0; i < new_size; i++) {
                int from = (HEAD + i) % CAPACITY;
                new_data[i] = DATA[from];
            }
            delete[] DATA;
            DATA = new_data;
            CAPACITY = new_capacity;
            HEAD = 0;
            TAIL = new_size % new_capacity;
            SIZE = new_size;
        }
};

#endif //BUFFER_H
```

Создан класс Buffer, который представляет собой кольцевой буфер. Этот буфер хранит элементы типа *T*, и его емкость задается при создании экземпляра класса. Класс Buffer предоставляет ряд методов для добавления и удаления элементов, а также для доступа к элементам и итераторам.

Метод push_back добавляет новый элемент в конец буфера, а метод push_front добавляет новый элемент в начало буфера. Если буфер заполнен, будет сгенерировано исключение std::out_of_range. Методы pop_back и pop_front

удаляют элементы из конца и начала буфера соответственно. Если буфер пуст, будет сгенерировано исключение std::out_of_range.

Метод at позволяет получить элемент по его индексу в буфере, а операторы [] позволяют получить элемент с помощью оператора доступа к массиву. Если указанный индекс находится за пределами диапазона индексов буфера, будет сгенерировано исключение std::out_of_range.

Класс Buffer также определяет вложенный класс Iterator, который предоставляет итераторы для доступа к элементам буфера. Этот итератор может быть использован для прохода по буферу с помощью стандартных алгоритмов STL.

File “main.cpp”:

```
#include <iostream>
#include "buffer.h"
using namespace std;

int main() {
    cout << "Welcome to my ring buffer!" << endl;
    int choice, value, index, start_capacity;
    cout << "INPUT CAPACITY: ";
    cin >> start_capacity;
    Buffer<int> buffer(start_capacity);

    while (true) {
        cout << "--------------------" << endl;
        cout << "What do you want to do?" << endl;
        cout << "1. Insert at the end" << endl;
        cout << "2. Insert at the beginning" << endl;
        cout << "3. Delete from the end" << endl;
        cout << "4. Delete from the beginning" << endl;
        cout << "5. Insert at a specific position" << endl;
        cout << "6. Delete from a specific position" << endl;
        cout << "7. Access the end" << endl;
        cout << "8. Access the beginning" << endl;
        cout << "9. Access by index" << endl;
        cout << "10. Change capacity" << endl;
        cout << "11. Print buffer" << endl;
        cout << "0. Exit" << endl;
        cout << "CHOICE: ";
        cin >> choice;
        cout << "--------------------" << endl;

        switch (choice) {
            case 1:
                cout << "Enter a value to insert: ";
                cin >> value;
                buffer.push_back(value);
                break;

            case 2:
                cout << "Enter a value to insert: ";
                cin >> value;
                buffer.push_front(value);
                break;

            case 3:
                buffer.pop_back();
                cout << "The element from the end is removed successfully" <<
```

```
endl;
                break;

            case 4:
                buffer.pop_front();
                cout << "The element from the beginning is removed
successfully" << endl;
                break;

            case 5:
                std::cout << "Enter the value to insert: ";
                std::cin >> value;
                std::cout << "Enter the index to insert at: ";
                std::cin >> index;
                try {
                    Buffer<int>::Iterator it = buffer.begin() + index;
                    buffer.insert(it, value);
                    std::cout << "Element inserted successfully" <<
std::endl;
                }
                catch (std::exception &e) {
                    std::cout << e.what() << std::endl;
                }
                break;

            case 6:
                std::cout << "Enter the index to remove: ";
                std::cin >> index;
                try {
                    Buffer<int>::Iterator it = buffer.begin() + index;
                    buffer.remove(it);
                    std::cout << "Element removed successfully" << std::endl;
                }
                catch (std::exception &e) {
                    std::cout << e.what() << std::endl;
                }
                break;

            case 7:
                try {
                    cout << "The last value in the buffer is: " <<
buffer.at(buffer.size() - 1) << endl;
                }
                catch (const out_of_range &e) {
                    cerr << e.what() << endl;
                }
                break;

            case 8:
                try {
                    cout << "The first value in the buffer is: " <<
buffer.at(0) << endl;
                }
                catch (const out_of_range &e) {
                    cerr << e.what() << endl;
                }
                break;

            case 9:
                cout << "Enter an index to access: ";
                cin >> index;
                try {
                    cout << "The value at index " << index << " is: " <<
buffer.at(index) << endl;
```

```
                }
                catch (const out_of_range &e) {
                    cerr << e.what() << endl;
                }
                break;

            case 10:
                cout << "Enter a new capacity: ";
                cin >> value;
                buffer.resize(value);
                break;

            case 11:
                std::cout << "Buffer contents: ";
                for (int i = 0; i < buffer.size(); i++) {
                    std::cout << buffer[i] << " ";
                }
                std::cout << std::endl;
                break;

            case 0:
                return 0;

            default:
                cout << "Invalid choice!!!" << endl;
                break;
        }
    }
    return 0;
}
```

В мэйне я решил создать консольную менюшку для более удобного использования буфера и демонстрации работоспособности всего требуемого функционала.

Создается объект типа Buffer<int>, используя конструктор, который принимает начальную ёмкость start_capacity, введенную пользователем.

Затем запускается бесконечный цикл, который выводит на экран список возможных действий с буфером, ожидает ввода пользователем номера выбранного действия choice и выполняет соответствующую операцию, используя switch.

Каждое действие соответствует одной из функций класса Buffer, например push_back, push_front, pop_back, pop_front, insert, remove, at, resize.

Внутри оператора switch() есть блоки try-catch, которые обрабатывают исключения, возникающие при попытке доступа к несуществующим элементам буфера или при попытке изменения его размера на отрицательное значение.

Цикл продолжается до тех пор, пока пользователь не выберет действие 0 - "выход" из программы.