Лабораторная работа №4

Выполнила: Аниськова Я.Ю.

Группа: 23ПИнж1з

**Вариант №1**

**Задание 1**. Реализовать структуры данных: стек, очередь, дек - используя:

- варианты 1,4,5,8,9,12,13 - массивы

- варианты 2,3,6,7,10,11,14,15 - связные списки.

**Реализация Стека**

#include <iostream>

#include <stdexcept>

class Stack {

private:

int\* arr;

int capacity;

int top;

public:

Stack(int size) : capacity(size), top(-1) {

arr = new int[capacity];

}

~Stack() {

delete[] arr;

}

void push(int value) {

if (top == capacity - 1) {

throw std::overflow\_error("Stack overflow");

}

arr[++top] = value;

}

int pop() {

if (top == -1) {

throw std::underflow\_error("Stack underflow");

}

return arr[top--];

}

int peek() const {

if (top == -1) {

throw std::underflow\_error("Stack is empty");

}

return arr[top];

}

bool isEmpty() const {

return top == -1;

}

int size() const {

return top + 1;

}

};

int main() {

Stack stack(5);

stack.push(10);

stack.push(20);

stack.push(30);

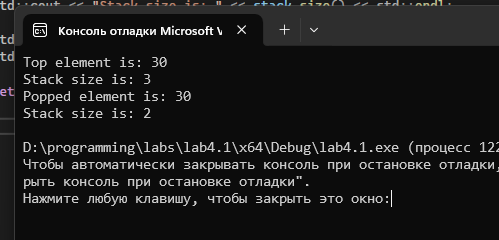
std::cout << "Top element is: " << stack.peek() << std::endl;

std::cout << "Stack size is: " << stack.size() << std::endl;

std::cout << "Popped element is: " << stack.pop() << std::endl;

std::cout << "Stack size is: " << stack.size() << std::endl;

return 0;

}

**Реализация Очереди**

#include <iostream>

#include <stdexcept>

class Queue {

private:

int\* arr;

int capacity;

int front;

int rear;

int count;

public:

Queue(int size) : capacity(size), front(0), rear(-1), count(0) {

arr = new int[capacity];

}

~Queue() {

delete[] arr;

}

void enqueue(int value) {

if (count == capacity) {

throw std::overflow\_error("Queue overflow");

}

rear = (rear + 1) % capacity;

arr[rear] = value;

count++;

}

int dequeue() {

if (isEmpty()) {

throw std::underflow\_error("Queue underflow");

}

int value = arr[front];

front = (front + 1) % capacity;

count--;

return value;

}

int peek() const {

if (isEmpty()) {

throw std::underflow\_error("Queue is empty");

}

return arr[front];

}

bool isEmpty() const {

return count == 0;

}

int size() const {

return count;

}

};

int main() {

Queue queue(5);

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

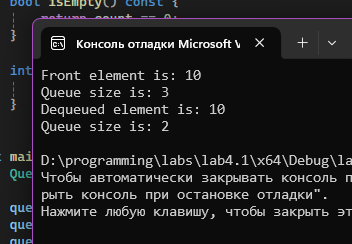
std::cout << "Front element is: " << queue.peek() << std::endl;

std::cout << "Queue size is: " << queue.size() << std::endl;

std::cout << "Dequeued element is: " << queue.dequeue() << std::endl;

std::cout << "Queue size is: " << queue.size() << std::endl;

return 0;

}

Реализация Дека

#include <iostream>

#include <stdexcept>

class Deque {

private:

int\* arr;

int capacity;

int front;

int rear;

int count;

public:

Deque(int size) : capacity(size), front(0), rear(size - 1), count(0) {

arr = new int[capacity];

}

~Deque() {

delete[] arr;

}

void insertFront(int value) {

if (isFull()) {

throw std::overflow\_error("Deque overflow");

}

front = (front - 1 + capacity) % capacity;

arr[front] = value;

count++;

}

void insertRear(int value) {

if (isFull()) {

throw std::overflow\_error("Deque overflow");

}

rear = (rear + 1) % capacity;

arr[rear] = value;

count++;

}

int deleteFront() {

if (isEmpty()) {

throw std::underflow\_error("Deque underflow");

}

int value = arr[front];

front = (front + 1) % capacity;

count--;

return value;

}

int deleteRear() {

if (isEmpty()) {

throw std::underflow\_error("Deque underflow");

}

int value = arr[rear];

rear = (rear - 1 + capacity) % capacity;

count--;

return value;

}

int getFront() const {

if (isEmpty()) {

throw std::underflow\_error("Deque is empty");

}

return arr[front];

}

int getRear() const {

if (isEmpty()) {

throw std::underflow\_error("Deque is empty");

}

return arr[rear];

}

bool isEmpty() const {

return count == 0;

}

bool isFull() const {

return count == capacity;

}

int size() const {

return count;

}

};

int main() {

Deque deque(5);

deque.insertRear(10);

deque.insertRear(20);

deque.insertFront(5);

std::cout << "Front element is: " << deque.getFront() << std::endl;

std::cout << "Rear element is: " << deque.getRear() << std::endl;

std::cout << "Deque size is: " << deque.size() << std::endl;

std::cout << "Deleted front element is: " << deque.deleteFront() << std::endl;

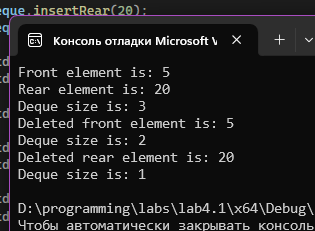
std::cout << "Deque size is: " << deque.size() << std::endl;

std::cout << "Deleted rear element is: " << deque.deleteRear() << std::endl;

std::cout << "Deque size is: " << deque.size() << std::endl;

return 0;

}



**Задание 2.**Используя реализацию из Задания 1 Вашего варианта, смоделировать следующие процессы (по вариантам):

***Вариант 1.*** Приложение, моделирующее работу конвейера, по производству салатов:

     - имеется склад продуктов, состоящих из 3 больших контейнеров (3 стека), в которые периодически доставляют 3 вида продукта по отдельности в каждый (по несколько единиц каждого продукта).  
     - периодически на склад приезжают 5 роботов, образуя очередь, и, если  в наличии имеются все 3 продукта, первый робот в очереди забирает по 1 ед. продукта из каждой коробки, смешивает их в салат, упаковывает и направляет в сортировочный цех. После доставки в сортировочный цех робот возвращается обратно в конец очереди.  
     - сортировочный цех представляет из себя ленту (дек), где в любой конец складываются упакованные салаты. С 2 концов ленты стоят 2 робота, которые готовые салаты складывают в машину (удаляют из дека по 1 единице салата), которая увозит продукцию в магазин.

#include <iostream>

#include <queue>

#include <deque>

#include <stack>

#include <thread>

#include <chrono>

#include <mutex>

std::mutex mtx;

class Stack {

private:

std::stack<int> stack;

public:

void push(int value) {

std::lock\_guard<std::mutex> lock(mtx);

stack.push(value);

}

bool pop() {

std::lock\_guard<std::mutex> lock(mtx);

if (stack.empty()) {

return false;

}

stack.pop();

return true;

}

bool isEmpty() {

std::lock\_guard<std::mutex> lock(mtx);

return stack.empty();

}

};

class Queue {

private:

std::queue<int> queue;

public:

void enqueue(int value) {

std::lock\_guard<std::mutex> lock(mtx);

queue.push(value);

}

bool dequeue() {

std::lock\_guard<std::mutex> lock(mtx);

if (queue.empty()) {

return false;

}

queue.pop();

return true;

}

bool isEmpty() {

std::lock\_guard<std::mutex> lock(mtx);

return queue.empty();

}

};

class Deque {

private:

std::deque<int> deque;

public:

void insertFront(int value) {

std::lock\_guard<std::mutex> lock(mtx);

deque.push\_front(value);

}

void insertRear(int value) {

std::lock\_guard<std::mutex> lock(mtx);

deque.push\_back(value);

}

bool deleteFront() {

std::lock\_guard<std::mutex> lock(mtx);

if (deque.empty()) {

return false;

}

deque.pop\_front();

return true;

}

bool deleteRear() {

std::lock\_guard<std::mutex> lock(mtx);

if (deque.empty()) {

return false;

}

deque.pop\_back();

return true;

}

bool isEmpty() {

std::lock\_guard<std::mutex> lock(mtx);

return deque.empty();

}

};

void supplyProducts(Stack& container1, Stack& container2, Stack& container3) {

while (true) {

std::this\_thread::sleep\_for(std::chrono::seconds(1));

container1.push(1);

container2.push(1);

container3.push(1);

std::cout << "Supplied products to the containers." << std::endl;

}

}

void produceSalads(Stack& container1, Stack& container2, Stack& container3, Queue& robotQueue, Deque& sortingConveyor) {

while (true) {

std::this\_thread::sleep\_for(std::chrono::seconds(1));

if (!container1.isEmpty() && !container2.isEmpty() && !container3.isEmpty() && !robotQueue.isEmpty()) {

container1.pop();

container2.pop();

container3.pop();

robotQueue.dequeue();

sortingConveyor.insertRear(1);

std::cout << "Produced a salad and sent it to the sorting conveyor." << std::endl;

robotQueue.enqueue(1);

}

}

}

void sortSalads(Deque& sortingConveyor) {

while (true) {

std::this\_thread::sleep\_for(std::chrono::seconds(1));

if (!sortingConveyor.isEmpty()) {

sortingConveyor.deleteFront();

std::cout << "Sorted a salad and sent it to the shop." << std::endl;

}

}

}

int main() {

Stack container1, container2, container3;

Queue robotQueue;

Deque sortingConveyor;

// Initialize robot queue

for (int i = 0; i < 5; ++i) {

robotQueue.enqueue(i);

}

// Start threads for supplying products, producing salads, and sorting salads

std::thread supplierThread(supplyProducts, std::ref(container1), std::ref(container2), std::ref(container3));

std::thread producerThread(produceSalads, std::ref(container1), std::ref(container2), std::ref(container3), std::ref(robotQueue), std::ref(sortingConveyor));

std::thread sorterThread(sortSalads, std::ref(sortingConveyor));

// Join threads

supplierThread.join();

producerThread.join();

sorterThread.join();

return 0;

}

