МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
**«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)**

Институт информационных технологий, математики и механики

Направление подготовки: «Программная инженерия»

ОТЧЕТ
по лабораторной работе

ОЧЕРЕДЬ

Выполнил: студент группы
3822Б1ПР2

_____ М.В.Фёдоров
Подпись

Проверила:

_____ Я.В. Силенко
Подпись

Нижний Новгород 2023

Содержание

Введение

Очередь — это структура данных, которая позволяет хранить и управлять набором элементов. В программировании очереди широко используются для решения различных задач, таких как управление процессами, планирование задач, организация взаимодействия между компонентами программного обеспечения и так далее.

Очереди широко используются в различных областях, включая:

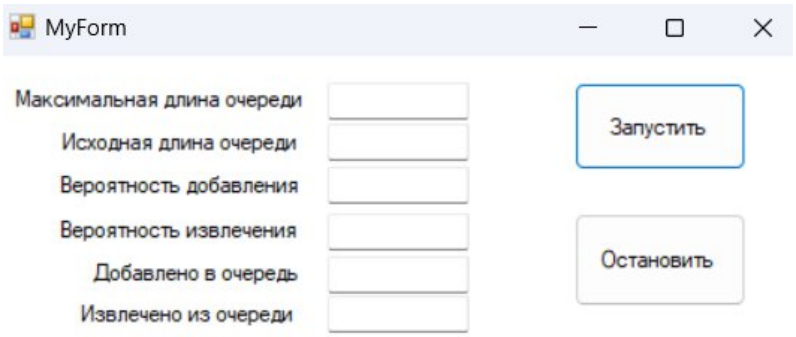
1. Разработку программного обеспечения: Очереди используются для управления процессами и потоками данных в приложениях, таких как веб-серверы, базы данных, системы управления задачами и т.д.
2. Мобильные приложения: Очереди используются для управления задачами и уведомлениями в мобильных приложениях, таких как социальные сети, мессенджеры, игры и т.д.
3. Обработку данных: Очереди используются для обработки больших объемов данных, таких как логи, данные сенсоров и т.д.
4. Облачные вычисления: Очереди используются для управления задачами и ресурсами в облачных вычислениях, таких как обработка запросов, балансировка нагрузки и т.д.
5. Искусственный интеллект: Очереди используются для управления задачами и обработки данных в системах искусственного интеллекта, таких как машинное обучение, нейронные сети и т.д.

Постановка задачи

Реализовать шаблонный класс Queue и создать визуальное приложение для имитационного моделирования работы очереди. На вход программе должны подаваться максимальный размер очереди, текущий размер очереди (количество элементов в очереди), вероятность добавления элемента в очередь и вероятность извлечения элемента из очереди.

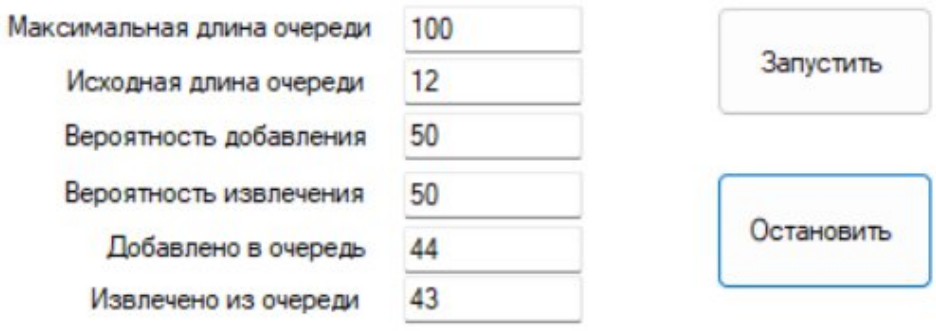
Руководство пользователя

При запуске программы на экране появляется визуальная форма MyForm с элементами управления моделью:



Для запуска программы с имитационным моделированием работы очереди пользователь должен ввести в соответствующие поля максимальную длину очереди, исходную длину очереди и вероятности добавления и извлечения элемента из очереди. В поля с вероятностями добавления и извлечения элемента требуется ввести число от 1 до 100 (вероятность в процентах).

После ввода указанных данных программа начинает моделировать работу очереди, а в двух нижних полях выводится информация о количестве добавленных и извлечённых элементов.



Руководство программиста

Описание структуры программы

Программа состоит из следующих интересующих нас файлов:

1) Queque.h – файл с реализацией шаблонного класса TQueue;
2) MyForm.h – файл с реализацией визуального интерфейса нашей программы;
3) MyForm.h [конструктор] – форма для визуального редактирования;

Описание структур данных

Class TQueue:

Поля:

int size – длина очереди;

int start – начало очереди;

int end – конец очереди;

int count – текущее количество элементов в очереди;

T* mas – массив элементов очереди;

Конструкторы и деструктор:

TQueue(int size = 0);

TQueue(TQueue <T>& q);

~TQueue();

Методы:

void Push(T a) – добавляет элемент в конец очереди;

T Get() – извлекает элемент из начала очереди и возвращает его:

bool IsFull() – проверка на полную очередь;

bool IsEmpty() – проверка на пустую очередь;

int GetSize() – возвращает текущее количество элементов в очереди;

int GetHead() – возвращает индекс начала очереди;

int GetMaxSize() – возвращает максимальный размер очереди;

Операторы:

friend istream& operator>>(istream& is, TQueue<T>& q) – перегрузка оператора ввода;

friend ostream& operator<<(ostream& os, const TQueue<T>& q) – перегрузка оператора вывода;

И отдельно реализация функций для отрисовки фигуры DrawQueue() и Clean() из MyForm.h:

```
void DrawQueue() {
	int start = 360 * pQueue->GetHead() / pQueue->GetMaxSize();
	int finish = 360 * pQueue->GetSize() / pQueue->GetMaxSize()
	gr->DrawArc(BlackPen, CenterX, CenterY, Width, Height, start, finish);
}
void Clean() {
	gr->DrawArc(WhitePen, CenterX, CenterY, Width, Height,0,360);
}
```

Описание алгоритмов

Очередь (англ. *queue*) — это структура данных, добавление и удаление элементов в которой происходит путём операций *push* и *pop* соответственно. Притом первым из очереди удаляется элемент, который был помещен туда первым, то есть в очереди реализуется принцип «первым вошел — первым вышел» (англ. *first-in, first-out — FIFO*). У очереди имеется **голова** (англ. *head*) и **хвост** (англ. *tail*). Когда элемент ставится в очередь, он занимает место в её хвосте. Из очереди всегда выводится элемент, который находится в ее голове. Очередь поддерживает следующие операции:

- IsEmpty — проверка очереди на наличие в ней элементов,
- push (запись в очередь) — операция вставки нового элемента,
- pop (снятие с очереди) — операция удаления нового элемента,

- GetCount — операция получения количества элементов в очереди.

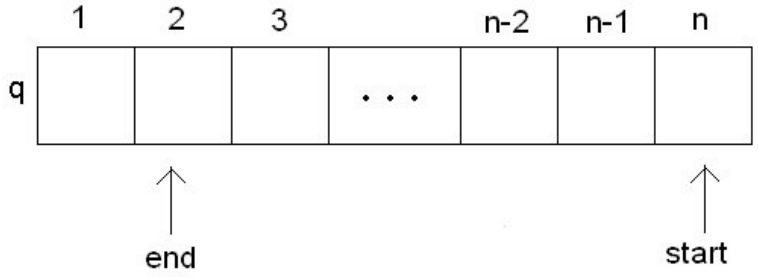


Рис.1

Способы реализации очереди

Первый способ представляет очередь в виде массива и двух целочисленных переменных start и end (см. рис. 1).

Обычно start указывает на голову очереди, end — на элемент, который заполнится, когда в очередь войдёт новый элемент. При добавлении элемента в очередь в q[end] записывается новый элемент очереди, а end уменьшается на единицу. Если значение end становится меньше 1, то мы как бы циклически обходим массив, и значение переменной становится равным n. Извлечение элемента из очереди производится аналогично: после извлечения элемента q[start] из очереди переменная start уменьшается на 1. С такими алгоритмами одна ячейка из n всегда будет незанятой (так как очередь с n элементами невозможно отличить от пустой), что компенсируется простотой алгоритмов.

Преимущества данного метода: возможна незначительная экономия памяти по сравнению со вторым способом; проще в разработке.

Недостатки: максимальное количество элементов в очереди ограничено размером массива. При его переполнении требуется перевыделение памяти и копирование всех элементов в новый массив.

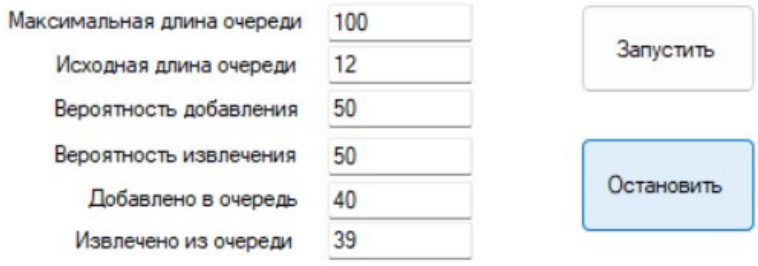
Второй способ основан на работе с динамической памятью. Очередь представляется в качестве линейного списка, в котором добавление/удаление элементов идет строго с соответствующих его концов.

Преимущества данного метода: размер очереди ограничен лишь объёмом памяти.

Недостатки: сложнее в разработке; требуется больше памяти; при работе с такой очередью память сильнее фрагментируется; работа с очередью несколько медленнее.

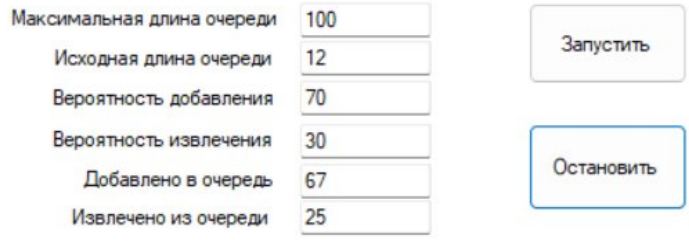
Результаты

При первом замере введём максимальную длину очереди, равной 100 элементам, текущую длину очереди равной 12, а вероятность добавления элемента в очередь будет равна вероятности извлечения элемента из очереди (50/50). Поставим таймер на 10 секунд и посмотрим, сколько элементов за это время было добавлено в и извлечено из очереди:



Как видим из результатов работы программы, количество элементов, добавленных в очередь, практически равно количеству элементов, извлечённых из очереди (40 против 39). Погрешность выполнения подсчётов есть следствие работы генератора случайных значений.

Теперь немного изменим условие: пусть вероятность добавления элемента в очередь равна 70%, а вероятность извлечения элемента из очереди – 30%.



Как видим из результатов работы программы, теперь количество элементов, добавленных очередь, значительно больше числа элементов, извлечённых из очереди (67 против 25). То есть очередь быстрее заполняется элементами, чем опустошается. Но переполнение очереди не произойдёт: в нашей программе циклическая очередь, и даже если текущий размер очереди достигнет её максимального размера, то после извлечения элемента из полной очереди в неё, соответственно, добавится другой элемент.

Заключение

Мною реализован шаблонный класс Queue, а также создано визуальное приложение для имитационного моделирования работы очереди. Результаты работы программы показали, что она работает корректно, погрешность генератора случайных значений в пределе допустимых значений.

Литература

1) Что такое очередь и способы её реализации: https://ru.wikipedia.org/wiki/%D0%9E%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)
2) Понятие очереди: https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C#:~:text=%D0%9E%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C%20(%D0%B0%D0%BD%D0%B3%D0%BB.,first%2Dout%20%E2%80%94%20FIFO).
3) Лекции ННГУ по курсу «Алгоритмы и структуры данных»

Приложение

```
#pragma once
#include <iostream>
#include <istream>
#ifndef TQUEQUE_H
#define TQUEQUE_H
#endif // !TQUEQUE_H

using namespace std;

template <class T>
class TStack {
protected:
	int size;
	int top;
	T* mas;
public:
	TStack() {
		size = 0;
		top = -1;
		mas = nullptr;
	}
	TStack(int _size = 0) {
		if (_size < 0) throw "error";
		top = -1;
		size = _size;
		mas = new T[size];
		for (int i = 0; i < size; i++) mas[i] = 0;
```

```
}
//TStack(int _size = 10, int _top = 0, T* pMem);

//TStack(int _size, int _top);
TStack(TStack<T>& stack) {
	top = stack.top;
	size = stack.size;
	mas = new T[size];
	for (int i = 0; i < size; i++) mas[i] = stack.mas[i];
}
~TStack() { delete mas; }

int GetSize()
{
	return top + 1;
}

T GetTop()
{
	if (top < 0) throw "error";
	return mas[top];
}

T Pop() {
	if (IsEmpty()) {
		throw "Stack is empty";
	}
	return mas[top--];
	//return mas[top--];
```

```
}
void Push(double elem) {
	if (IsFull()) {
		throw - 1;
	}
	//pMem[top] = elem;
	//top++;
	mas[++top] = elem;
}
bool IsEmpty() { return top + 1 == 0; }



bool IsFull() { return top + 1 == size; }

void Clear() { for (int i = 0; i < size; i++) mas[i] = 0; }

T Get() {
	if (IsEmpty()) {
		throw - 1;
	}
	top--;
	return mas[top];
}

TStack& operator=(const TStack<T>& stack) {
	if (&stack == this) return *this;
	size = stack.size;
	top = stack.top;
	delete[] mas;
```

```
            mas = new T[size];
            for (size_t i = 0; i < size; i++) mas[i] = stack.mas[i];
            return *this;
        }
};

template <class T>
class TQueue{
protected:
        int size;
        int start;
        int end;
        int count;
        T* mas;

public:
        TQueue(int size = 0);
        TQueue(TQueue <T>& q);
        ~TQueue();

        void Push(T a);
        T Get();

        bool IsFull();
        bool IsEmpty();

        int GetSize();
        int GetHead();
        int GetMaxSize();
```

```
    friend istream& operator>>(istream& is, TQueue<T>& q);
    friend ostream& operator<<(ostream& os, const TQueue<T>& q);

};



template<class T>
TQueue<T>::TQueue(int size) {
    this->size = size;
    start = 0;
    end = 0;
    count = 0;
    mas = new T[size]{};
}

template<class T>
TQueue<T>::TQueue(TQueue<T>& q) {
    size = q.size;
    start = q.start;
    end = q.end;
    count = q.count;

    mas = new T[size]{};
    for (int i = 0; i < size; ++i) {
        mas[i] = q.mas[i];
    }
```

```
}

template<class T>
TQueue<T>::~TQueue() {
	delete[] mas;
	size = 0;
	start = 0;
	end = 0;
	count = 0;
}

template<class T>
void TQueue<T>::Push(T a)
{
	if (!IsFull()) {
		mas[end] = a;
		end = (end + 1) % size;
		++count;
	}
	else {
		cout << "Queue is full\n";
	}
}

template<class T>
T TQueue<T>::Get()
{
	T res;
	if (!IsEmpty()) {
```

```
		res = mas[start];
		start = (start + 1) % size;
		--count;
	}
	else {
		cout << "Queue is empty\n";
		res = T();
	}
	return res;
}

template<class T>
bool TQueue<T>::IsFull() {
	return count == size;
}

template<class T>
bool TQueue<T>::IsEmpty() {
	return count == 0;
}

template<class T>
int TQueue<T>::GetSize() {
	return count;
}

template<class T>
int TQueue<T>::GetHead() {
	return start;
```

```
}

template<class T>
int TQueue<T>::GetMaxSize() {
	return size;
}

template<class T>
istream& operator>>(istream& is, TQueue<T>& q) {
	for (int i = 0; i < q.size; ++i) {
		is >> q.mas[i];
	}
	return is;
}

template<class T>
ostream& operator<<(ostream& os, const TQueue<T>& q) {
	for (int i = 0; i < q.size; ++i) {
		os << q.mas[i] << " ";
	}
	return os;
}
```