Министерство образования и науки Российской Федерации

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**Разработка структуры данных очередь и применение её для решения практической задачи**

**Выполнил**:студент группы 381606-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимакин Н.Е.

Подпись

**Проверил**: к.ф.-м.н., доц.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Баркалов К.А.

Подпись

Нижний Новгород 2017

**Введение**

Очередь – это структура данных, представляющая собой список элементов и организованная по принципу: «первый пришёл – первый ушёл», то есть добавление элемента происходит только в конец, а удаление – только из начала. Очередь не только часто встречается в повседневной жизни (в магазинах, на почте, в банках), но ещё с её помощью решаются многие прикладные задачи. Одна из таких задач и будет рассмотрена как пример использования этой структуры данных.

**Постановка задачи**

Реализовать структуру данных очередь как шаблонный класс, написать к нему тесты, подтверждающие его работоспособность, и тестовую программу – имитацию работы многопроцессорной системы в виде приложения с графическим интерфейсом.

**Описание структуры программы**

Программа содержит в себе 3 проекта:

1. Queue lab – реализация очереди
   * TQueue.h – объявление и реализация класса TQueue
   * main.cpp – файл с функцией main()
2. Test – тесты для очереди
   * gtest.h – заголовочный файл библиотеки Google Test
   * gtest-all.cc – реализация библиотеки Google Test
   * test\_main.cpp – файл, запускающий все тесты
   * test\_queue.cpp – тесты для класса TQueue
3. ClusterImmitation – приложение с графическим интерфейсом
   * Structs.h – файл с объявлением структур для реализации алгоритма
   * MyForm.h – объявление и реализация формы для графического приложения
   * MyFoem.cpp – файл для запуска приложения

T – шаблон

**Класс TQueue**

Поля:

int first – номер первого элемента

int last – номер следующего за последним элемента

int size – текущий размер очереди

int maxsize – максимальный размер очереди

T\* ring – память для хранения элементов

Методы:

TQueue(int \_maxsize = 10) – конструктор по умолчанию

TQueue(const TQueue& q) – конструктор копирования

~TQueue() - деструктор

int isempty() – проверка на пустоту

int isfull() – проверка на полноту

void push(const T elem) – добавить элемент

T pop() – удалить и получить элемент

T getLastElem() – получить последний элемент

int getFirst() – получить номер первого элемента

int getSize() – получить текущий размер

int getMaxsize() – получить максимальный размер

T top() – получить первый элемент

int getLast() – получить номер следующего за последним элемента

int operator==(const TQueue& q) const – перегрузка операции сравнения

int operator!=(const TQueue& q) const – перегрузка операции не равно

**Описание алгоритмов**

**Класс TQueue**

Главная идея реализации очереди состоит в её представлении кольцевым буфером, чтобы не перепаковывать массив элементов на каждой итерации программы. Для этого необходимо индексировать массив таким образом, чтобы следующим элементом для элемента с номером maxsize-1 был нулевой. Это легко достигается, если обращаться к элементу по остатку от деления номера на максимальный размер (тогда элемент с номером maxsize станет нулевым, maxsize+1 – первым и т.д.)

TQueue(int \_maxsize = 10) {если \_maxsize <= 0, выбросить исключение. Максимальному размеру присвоить \_maxsize. Выделить память объёмом максимального размера. Номеру первого элемента присвоить 0, последнего - -1. Текущему размеру присвоить 0}

TQueue(const TQueue& q) {максимальному размеру присвоить максимальный размер q. Выделить память объёмом максимального размера. Номеру первого элемента присвоить номер первого элемента q, последнего – номер последнего q. Текущему размеру присвоить текущий размер q. Присвоить каждому элементу соответствующий из q}

~TQueue() {освободить память}

int isempty() {если текущий размер 0, то вернуть 1, иначе – 0}

int isfull() {если текущий размер равен максимальному, вернуть 1, иначе – 0}

void push(const T elem) {если очередь полная, то выбросить исключение. Последнему присвоить остаток от деления последнего+1 на максимальный размер. Последнему элементу присвоить значение elem. Увеличить размер на 1}

T pop() {если очередь пустая, выбросить исключение. Запомнить номер первого элемента во временной переменной. Первому присвоить остаток от деления первого на максимальный размер. Уменьшить размер на 1. Вернуть элемент с номером, который запомнили ранее}

T getLastElem() {вернуть элемент с последним номером}

int getFirst() {вернуть номер первого элемента}

int getSize() {вернуть текущий размер}

int getMaxsize() {вернуть максимальный размер}

T top() {если очередь пустая, выбросить исключение. Вернуть элемент с первым номером}

int getLast() {вернуть номер следующего за последним элемента}

int operator==(const TQueue& q) const {если максимальные размеры, текущие размеры, первые номера или последние номера не равны, то вернуть 0. Если хотя бы один из элементов с первого до последнего не равен, вернуть 0. Вернуть 1}

int operator!=(const TQueue& q) const {вернуть отрицание от проверки на равенство \*this и q}

**Приложение с графическим интерфейсом**

Для реализации

**Результаты**

**Вывод**

**Литература**

Брайан Керниган, Деннис Ритчи «Язык программирования Си»

Брюс Эккель «Философия С++. Введение в стандартный С++»

Стивен Прата «Язык программирования С++. Лекции и упражнения»

**Приложение**