Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Нижегородский государственный университет им. Н.И. Лобачевского»

Факультет вычислительной математики и кибернетики

Отчет по лабораторной работе

Очередь

Выполнил: Назаров Н.С. 381906-2

Проверил:

Нижний Новгород

2020 г.

Содержание

1. Введение 3
2. Постановка задачи 4
3. Руководство пользователя 5
4. Руководство программиста 6

**4.1** Класс Queue 7

1. Заключение 9
2. Приложение 10

Введение

Очередь представляет собой линейный список данных, доступ к которому осуществляется по принципу «первый вошел, первый вышел» (иногда сокращенно его называют методом доступа FIFO). Элемент, который был первым поставлен в очередь, будет первым получен из нее. Элемент, поставленный в очередь вторым, при поиске будет получен также вторым и т. д. Существует два способа реализации очереди — статический и динамический. При статической реализации элементы очереди хранятся в обычном массиве. Память под массив выделяется сразу и не изменяется по мере заполнения очереди. Преимуществом данного способа является скорость работы с очередью, т.к. элементы очереди располагаются в памяти друг за другом. Недостатком является возможность переполнения очереди при слишком большом количестве добавленных элементов. При динамической реализации каждый элемент очереди состоит из двух частей — информационной части и указателя на следующий элемент очереди. При таком способе реализации элементы располагаются в памяти произвольным образом. Преимуществом данного способа является то, что размеры очереди ограничены лишь размерами памяти и пустые элементы не хранятся в очереди, а освобождаются из памяти. Недостатком является медленная скорость работы с элементами при большом их количестве.

Постановка задачи

Очередь – схема запоминания информации, при которой каждый вновь поступающий ее элемент занимает крайнее положение (конец очереди). При выдаче информации из очереди выдается элемент, расположенный в очереди первым (начало очереди), а оставшиеся элементы продвигаются к началу; следовательно, элемент, поступивший первым, выдается первым.

Руководство пользователя

Queue(int size = 1); // Конструктор

Queue(Queue<ValType>& v); // Конструктор копирования

~Queue(); // Деструктор

Queue<ValType>& operator=(Queue<ValType>& v); // Оператор копирования

void PushQueue(ValType temp); // Добавляет элемент в конец очереди

ValType GetQueue(); // Вытягивает первый элемент из очереди

template <class ValType1>

friend ostream& operator<< (ostream& ostr, const Queue<ValType1>& p); // Перегрузка операции сдвига

template <class ValType1>

friend istream& operator>> (istream& istr, Queue<ValType1>& p); // Перегрузка операции сдвига

int Length() {return length;} // Возвращает длину очереди

void MaxElement(int LengthQueue); // Находит максимальный элемент в очереди

Руководство программиста

Очередь - структура данных, представляющая собой список элементов, организованных по правилу FIFO (first in - first out, то есть первым пришёл - первым ушёл). Элемент добавляется в конец очереди, а извлечение элемента происходит из начала очереди. Рассмотрим его реализацию с помощью массива. Графически очередь может быть представлена так:



Рисунок 1. Динамическая структура данных «очередь»

Одной из особенностей очереди есть то, что в очереди часть последовательных элементов может быть занята на данный момент времени :

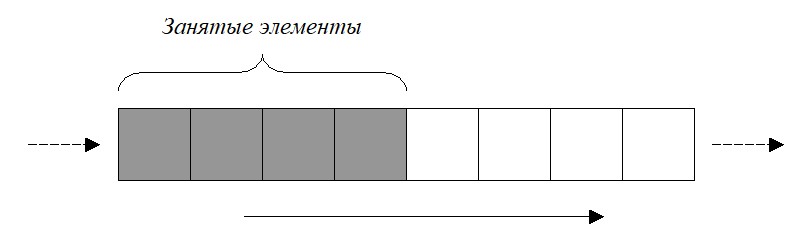


Рисунок 2. Очередь, в которой часть элементов занята на данный момент времени

Но когда указатель последней записи в очереди не пустой и он показывает на первую запись , то нашу очередь мы может представить в виде окружности:

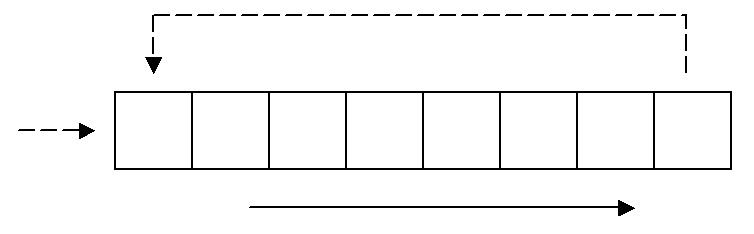


Рисунок 3. Общая схема структуры хранения вида кольцевой буфер

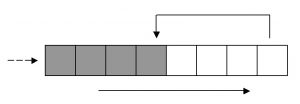


Рисунок 4. Кольцевая очередь. Случай когда часть элементов занята

Интерфейс класса Queue:

template <class ValType>

class Queue{

protected:

int length;

ValType\* pQueue;

int first;

int last;

int count;

public:

Queue(int size = 1);

Queue(Queue<ValType>& v);

~Queue();

Queue<ValType>& operator=(Queue<ValType>& v);

void PushQueue(ValType temp);

ValType GetQueue();

template <class ValType1>

friend ostream& operator<< (ostream& ostr, const Queue<ValType1>& p);

template <class ValType1>

friend istream& operator>> (istream& istr, Queue<ValType1>& p);

int Length() {return length;}

void MaxElement(int LengthQueue);

};

Рассмотрим реализацию некоторых методов:

Конструктор:

template <class ValType>

Queue<ValType>::Queue(int size = 1) {

if (size > 0) {

this->length = size;

pQueue = new ValType[length];

for (int i = 0; i < length; i++)

pQueue[i] = 0;

this->first = -1;

this->last = -1;

this->count = 0;

}

else

throw logic\_error("Input error: invalide value of Queue length in constructor");

}

Вытягивание первого элемента из очереди:

Для этого уменьшаем число элементов в очереди на один, и обрабатываем два случая: если количество элементов 0, то кидаем исключение, иначе увеличиваем голову на единицу и возвращаем предыдущий элемент от головы в массиве.

template<class ValType>

inline ValType Queue<ValType>::GetQueue(){

if (count == 0)

throw logic\_error("Input error: invalide value of Queue length in GetQueue");

ValType temp = pQueue[first];

first = (first + 1) % length;

--count;

return temp;

}

Добавление элемента в конец очереди:

Для этого обрабатываем два случая, если количество элементов больше либо равно длине массива, то кидаем исключение, иначе увеличиваем позицию конца, затем приписываем элемент в массив на позицию конца очереди и увеличиваем число элементов в очереди.

template<class ValType>

inline void Queue<ValType>::PushQueue(ValType temp){

if (count >= length)

throw logic\_error("Input error: invalide value of Queue length in PushQueue");

last = (last + 1) % length;

pQueue[last] = temp;

++count;

}

Заключение

В квалификационной работе мы попытались раскрыть более полно и наглядно понятие очереди. Сформировать и закрепить познавательный интерес к данной теме у учащихся. Выявлять и развивать творческие способности в использовании полученного навыка при программировании. Все поставленные цели и задачи реализованы в данной квалификационной работе. Кроме того, рассматривается более конкретно и полно каждая очередь в отдельности и показывается целесообразность использования очереди в тех или иных случаях. И последнее это руководство пользователя, в котором дается описания работы с демонстрационной программой.

Приложение

Код класса Queue:

[Queue.cpp](https://github.com/Nazarov69/-Stack-and-queue/blob/NewBranch/Queuelib/Queue.cpp" \o "Queue.cpp)

#include <math.h>

#include "Queue.h"

[Queue.h](https://github.com/Nazarov69/-Stack-and-queue/blob/NewBranch/Queuelib/Queue.h" \o "Queue.h)

#ifndef \_Queue\_

#define \_Queue\_

#include <iostream>

using namespace std;

template <class ValType>

class Queue{

protected:

int length;

ValType\* pQueue;

int first;

int last;

int count;

public:

Queue(int size = 1);

Queue(Queue<ValType>& v);

~Queue();

Queue<ValType>& operator=(Queue<ValType>& v);

void PushQueue(ValType temp);

ValType GetQueue();

template <class ValType1>

friend ostream& operator<< (ostream& ostr, const Queue<ValType1>& p);

template <class ValType1>

friend istream& operator>> (istream& istr, Queue<ValType1>& p);

int Length() { return length; }

void MaxElement(int LengthQueue);

};

template<class ValType>

inline void Queue<ValType>::MaxElement(int LengthQueue){

int tmp1 = 0;

int tmp2 = 0;

for (int i = 0; i < LengthQueue; i++)

PushQueue(rand());

cout << "queue = {";

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

tmp1 = GetQueue();

cout << tmp1;

if (i < LengthQueue - 1) cout << ", ";

if (i == 0) tmp2 = tmp1;

if (tmp1 > tmp2) tmp2 = tmp1;

}

cout << "} " << endl;

cout << "max element in queue: " << tmp2 << endl;

}

template <class T1>

ostream& operator<< (ostream& ostr, const Queue<T1> &p) {

for (int i = p.first; i < p.last; i++)

ostr << p.pQueue[i] << endl;

return ostr;

}

template <class ValType1>

istream& operator >> (istream& istr, Queue<ValType1> &p) {

int count;

istr >> count;

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

ValType1 temp;

istr >> temp;

p.PushQueue(temp);

}

return istr;

}

template <class ValType>

Queue<ValType>::Queue(int size = 1) {

if (size > 0) {

this->length = size;

pQueue = new ValType[length];

for (int i = 0; i < length; i++)

pQueue[i] = 0;

this->first = -1;

this->last = -1;

this->count = 0;

}

else

throw logic\_error("Input error: invalide value of Queue length in constructor");

}

template <class ValType>

Queue<ValType>::Queue(Queue<ValType>& v){

length = v.length;

first = v.first;

last = v.last;

count = v.count;

pQueue = new ValType[length];

for (int i = 0; i < count; i++)

pQueue[i] = v.pQueue[i];

}

template <class T>

Queue<T>::~Queue()

{

length = 0;

if (pQueue != 0)

delete [] pQueue;

pQueue = nullptr;

}

template <class ValType>

Queue<ValType>& Queue<ValType>::operator=(Queue<ValType>& v)

{

if (this == &v)

return \*this;

length = v.length;

first = v.first;

last = v.last;

count = v.count;

delete[] pQueue;

pQueue = new ValType[length];

for (int i = 0; i < count; i++)

pQueue[i] = v.pQueue[i];

return \*this;

}

template<class ValType>

inline void Queue<ValType>::PushQueue(ValType temp){

if (count >= length)

throw logic\_error("Input error: invalide value of Queue length in PushQueue");

last = (last + 1) % length;

pQueue[last] = temp;

++count;

}

template<class ValType>

inline ValType Queue<ValType>::GetQueue(){

if (count == 0)

throw logic\_error("Input error: invalide value of Queue length in GetQueue");

ValType temp = pQueue[first];

first = (first + 1) % length;

--count;

return temp;

}

#endif