Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

**Имитационное моделирование системы обслуживания потока заданий на ЭВМ (очереди)**

Выполнил:

студент ф-та ПРИН гр. 381908-4

Бутеску.В.В

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г

Нижний Новгород

2020 г.

Содержание

[Введение 3](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962758)

[Постановка задачи 4](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962759)

[Руководство программиста 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962761)

[Описание структур данных 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962762)

[Описание алгоритмов 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962763)

[Описание структуры программы 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962764)

[Заключение 7](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962765)

[Литература 8](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962766)

[Приложения 9](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962767)

# Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных Очередь. С этой целью в лабораторной работе изучаются различные варианты структуры хранения очереди и разрабатываются методы и программы решения задач с использованием очередей. В качестве области приложений выбрана тема эффективной организации выполнения потока заданий на вычислительных системах.

Очередь характеризуется таким порядком обработки значений, при котором вставка новых элементов производится в конец очереди, а извлечение – из начала. Подобная организация данных широко встречается в различных приложениях. В качестве примера использования очереди предлагается задача разработки системы имитации однопроцессорной ЭВМ. Рассматриваемая в рамках лабораторной работы схема имитации является одной из наиболее простых моделей обслуживания заданий в вычислительной системе и обеспечивает тем самым лишь начальное ознакомление с проблемами моделирования и анализа эффективности функционирования реальных вычислительных систем.

# Постановка задачи

Для вычислительной системы (ВС) с одним процессором и однопрограммным последовательным режимом выполнения поступающих заданий требуется разработать программную систему для имитации процесса обслуживания заданий в ВС. При построении 47 модели функционирования ВС должны учитываться следующие основные моменты обслуживания заданий:

• генерация нового задания;

• постановка задания в очередь для ожидания момента освобождения процессора;

• выборка задания из очереди при освобождении процессора после обслуживания очередного задания.

По результатам проводимых вычислительных экспериментов система имитации должна выводить информацию об условиях проведения эксперимента (интенсивность потока заданий, размер очереди заданий, производительность процессора, число тактов имитации) и полученные в результате имитации показатели функционирования вычислительной системы, в т.ч.

• количество поступивших в ВС заданий;

• количество отказов в обслуживании заданий из-за переполнения очереди;

• среднее количество тактов выполнения заданий;

• количество тактов простоя процессора из-за отсутствия в очереди заданий для обслуживания.

Показатели функционирования вычислительной системы, получаемые при помощи систем имитации, могут использоваться для оценки эффективности применения ВС; по результатам анализа показателей могут быть приняты рекомендации о целесообразной модернизации характеристик ВС (например, при длительных простоях процессора и при отсутствии отказов от обслуживания заданий желательно повышение интенсивности потока обслуживаемых заданий и т.д.).

**Руководство пользователя**

Пользователю нужно запустить файл main.exe.

Откроется консольное приложение для тестирования очереди.

Программа заполнит очередь числами и выведет их в консоль, так же выведет копирования очереди

Для повторного выполнения потребуется перезапустить программу.

# Руководство программиста

## **Описание структуры программы**

Реализует операции:

rotected:

int length;

T\* x;

int ind; //Индекс начала

int end; //Индекс конца

int count; //Количество

public:

TQueue(int size = 0);

TQueue(TQueue<T>& \_v);

~TQueue();

TQueue<T>& operator =(TQueue<T>& \_v);

void Push(T d); //Вставка элемента

T Get(); //Получение элемента

int Length();

int IsEmpty(void) const; // контроль пустоты

//Доп задания

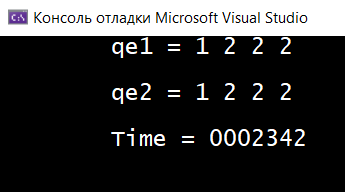
inline int min\_elem(); //Поиск минимального элемента

inline int max\_elem(); //Поиск максимального элемента

inline void file(); //Запись в файл

# Эксперименты

Конструктор копирования



# Заключение

Этап 1. Реализация программ поддержки очереди.

Этап 2. Реализация системы имитации обслуживания заданий.

Этап 3. Выполнение дополнительных заданий лабораторной работы.

# Литература

1. Столлингс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 896 с.: ил. — Парал. тит. англ.
2. Johnson M. Superscalar Microprocessor Design. — Englewood Cliff, New Jersey: Prentice Hall, 1991.
3. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
4. Stone H. High performance Computer Architecture. — Reading, MA: Addison-Wesley, 1993.
5. Tullsen D.M., Eggers S.J. Effective Cache Prefetching on a Bus-Based Multiprocessor. — ACM Transactions on Computer Systems, pp. 57-88, Feb 1995.
6. Chandra D., Guo F., Kim S., Solihin Y. Predicting inter-thread cache contention on a chip multi-processor architecture. — Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA), pp. 340–351, Feb 2005.
7. Press W., Teukolsky S., Vetterling W., Flannery B. Numerical Recipes in C. The Art of Scientific Computing. Second Edition. — Cambridge University Press, 1992.
8. Камаев А.М., Сиднев А.А., Сысоев А.В. Об одном подходе к анализу эффективности приложений // Труды 50-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук»: Часть I. Радиотехника и кибернетика. - М.: МФТИ, 2007.
9. Debugging and performance monitoring. Intel® 64 and IA-32 Architectures Software Developer’s Manual. Volume 3B: System Programming Guide, Part 2. May 2007. — [http://www.intel.com/products/processor/manuals/]
10. Юнаковский А.Д. Начала вычислительных методов для физиков. – Н. Новгород: ИПФ РАН, 2007.

# Приложения

Queue.h

#ifndef \_QUEUE\_

#define \_QUEUE\_

#include <iostream>

#include <fstream>

using namespace std;

template <class T>

class TQueue

{

protected:

int length;

T\* x;

int ind; //Индекс начала

int end; //Индекс конца

int count; //Количество

public:

TQueue(int size = 0);

TQueue(TQueue<T>& \_v);

~TQueue();

TQueue<T>& operator =(TQueue<T>& \_v);

void Push(T d); //Вставка элемента

T Get(); //Получение элемента

int Length();

int IsEmpty(void) const; // контроль пустоты

//Доп задания

inline int min\_elem(); //Поиск минимального элемента

inline int max\_elem(); //Поиск максимального элемента

inline void file(); //Запись в файл

template <class T1>

friend ostream& operator<< (ostream& ostr, const TQueue<T1>& A);

template <class T1>

friend istream& operator >> (istream& istr, TQueue<T1>& A);

};

#endif

Queue.cpp

#include "Queue.h"

template <class T1>

ostream& operator<< (ostream& ostr, const TQueue<T1>& A)

{

for (int i = A.ind; i < A.end; i++)

{

ostr << A.x[i] << endl;

}

return ostr;

}

template <class T1>

istream& operator >> (istream& istr, TQueue<T1>& A)

{

int count;

istr >> count;

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

T1 d;

istr >> d;

A.Push(d);

}

return istr;

}

template<class T>

inline TQueue<T>::TQueue(int size)

{

if (size > 0)

{

this->length = size;

x = new T[length];

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

{

x[i] = 0;

}

this->ind = 0;

this->end = 0;

this->count = 0;

}

else

{

throw logic\_error("ERROR");

}

}

template <class T>

TQueue<T>::TQueue(TQueue<T>& \_v)

{

length = \_v.length;

ind = \_v.ind;

end = \_v.end;

count = \_v.count;

x = new T[length];

for (int i = 0; i < length; i = i + 1)

{

x[i] = \_v.x[i];

}

}

template <class T>

TQueue<T>::~TQueue()

{

length = 0;

if (x != 0)

delete[] x;

x = 0;

}

template <class T>

TQueue<T>& TQueue<T>::operator =(TQueue<T>& \_v)

{

if (this == &\_v)

return \*this;

this->length = \_v.length;

if (x != NULL)

{

delete[] x;

}

this->x = new T[length];

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

x[i] = \_v.x[i];

this->ind = \_v.ind;

this->end = \_v.end;

this->count = \_v.count;

return \*this;

}

template<class T>

inline void TQueue<T>::Push(T d)

{

if (count >= length)

{

ind++;

count--;

}

x[end] = d;

end = (end + 1) % length;

count++;

}

template<class T>

inline T TQueue<T>::Get()

{

if (count == 0)

{

throw logic\_error("ERROR");

}

T d = x[ind];

ind = (ind + 1) % length;

return d;

}

template <class T>

int TQueue<T>::Length()

{

return length;

}

template<class T>

inline int TQueue<T>::IsEmpty(void) const

{

return count == 0;

}

//ДОП

template<class T>

inline int TQueue<T>::min\_elem()

{

int res = x[0];

for (int i = 1; i < length; i++)

{

if (x[i] < res)

{

res = x[i];

}

}

return res;

}

template<class T>

inline int TQueue<T>::max\_elem()

{

int res = x[0];

for (int i = 1; i < length; i++)

{

if (x[i] > res)

{

res = x[i];

}

}

return res;

}

template<class T>

inline void TQueue<T>::file()

{

ofstream outf("Data.txt");

if (!outf)

{

throw logic\_error ("Error file");

}

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

{

outf << x[i] << endl;

}

}