|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_Информатика и Системы Управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_Программное обеспечение ЭВМ и информационные технологии\_\_\_\_

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5**

**«ОБРАБОТКА ОЧЕРЕДЕЙ»**

Студент\_\_\_\_\_\_Чыонг Ван Хао\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*фамилия, имя, отчество*

Группа\_\_\_\_\_*ИУ7И-31Б\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

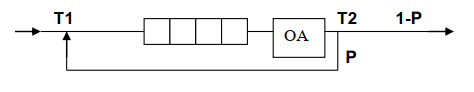
Вариант 5

Принял : Силантьева А. В.

**Описание условия задачи**

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и

очереди заявок.



Заявки поступают в "хвост" очереди по случайному закону с интервалом

времени Т1, равномерно распределенным от 0 до 6 единиц времени (е.в.). В ОА

они поступают из "головы" очереди по одной и обслуживаются также

равновероятно за время Т2 от 0 до 1 е.в. Каждая заявка после ОА с

вероятностью Р=0.8 вновь поступает в "хвост" очереди, совершая новый цикл

обслуживания, а с вероятностью 1-Р покидает систему. (Все времена –

вещественного типа). В начале процесса в системе заявок нет.

Смоделировать процесс обслуживания до ухода из системы первых 1000 заявок.

Выдавать после обслуживания каждых 100 заявок информацию о текущей и

средней длине очереди. В конце процесса выдать общее время моделирования и

количество вошедших в систему и вышедших из нее заявок, среднее время

пребывания заявки в очереди, время простоя аппарата, количество

срабатываний ОА. Обеспечить по требованию пользователя выдачу на экран

адресов элементов очереди при удалении и добавлении элементов.

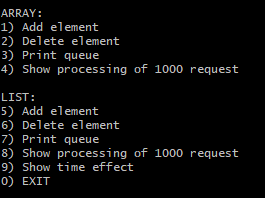
Проследить, возникает ли при этом фрагментация памяти.

**Техническое задание**

**Входные данные**

Программа для получения данных из вручную.

Целое число — номер пункта меню, который вызывает описанное в пункте  
действие.



1. Добавить элемент в очередь.
2. Удалить элемент.
3. Показать очередь.
4. Смоделировать процесс обслуживания заявок с выводом состояния очередей.
5. Добавить элемент. (для списка)
6. Удалить элемент. (для списка)
7. Показать очередь. (для списка)
8. Смоделировать процесс обслуживания заявок с выводом состояния очередей. (для списка)

9) Вывести эффективность.

0) выйти из программы

**Выходные данные**

1. Целые числа — текущие элементы очереди.

2. Адреса — адреса текущих элементов очереди или адреса свободных областей.

3. Временная характеристика — время работы функций.

4. Объемная характеристика — объем структуры данных.

**Функция программы**

Операции работы с очередью — добавление элемента, удаление элемента,

печать текущего состояния и адресов элементов, печать свободных областей,

моделирование процесса обслуживания 1000 заявок.

**Обращение к программе**

Программа запускается из терминала командой «./app.exe» в директории с  
программой.

**Возможные аварийные ситуации и ошибки пользователя**

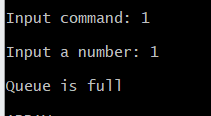
1. Ввести неправильный выбор.

На выходе сообщение: «ERROR\_COMMAND.»



2. Очередь переполнена.

На выходе сообщение: « Queue is full »



3. Удаление элемента из пустой очереди

На выходе сообщение: « Queue is empty!



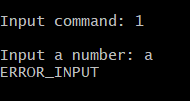
4. Вывод пустой очереди

На выходе сообщение: « Queue is empty! »



5. Неправильный ввод элемента

На выходе сообщение: « ERROR\_INPUT»



**Внутренняя структура данных**

Для реализации очереди при помощи массива используется структура:

typedef struct

{

    int queue[MAX\_ELEM];

    int head;

    int last;

    int len;

} array\_queue\_t;

Её поля:

*int queue[MAX\_ELEM]* — очередь, где *MAX\_ELEM* — её максимальныйразмер, равный 1000. *int head —* индекс начала очереди; *int last —* индекс конца очереди; *int len —* текущее число элементов;

Для реализации очереди при помощи односвязного списка используется  
структуры:

typedef struct node

{

    int elem;

    struct node \*next;

} node\_t;

Её поля:  
int elem — значение элемента;  
struct node \*next — указатель на следующий элемент;

typedef struct list\_queue\_t

{

    int len;

    node\_t \*head;

    node\_t \*last;

} list\_queue\_t;

Её поля:  
int len— текущее число элементов;  
node\_t \*head — указатель на начало очереди;  
node\_t \*last — указатель на конец очереди;

Для массива свободных областей используется структура:  
size\_t \*free\_elem — массив адресов;  
int count\_free— текущее число адресов;

**Функции программы**

void menu(void);

Описание: печать на экран параметров выполнения.

void message(int rc);

Описание: показать сообщение об ошибке

int input\_number(int \*elem);

Описание: введите 1 элемент.

**Функции для работы с очередью, реализованной при помощи массива:**

int array\_queue\_add(array\_queue\_t \*array, int elem);

Описание: добавить элемент в очередь.

int array\_queue\_pop(array\_queue\_t \*array);

Описание: Удалить элемент из очереди.

void print\_array\_queue(array\_queue\_t array);

Описание: Показать очередь.

void array\_queue\_process(void);

Описание: Смоделировать процесс обслуживания заявок с выводом состояния очередей.

**Функции для работы с очередью, реализованной при помощи односвязного  
списка, и с массивом свободных областей:**

node\_t \*create\_node(const int elem);

Описание: инициализировать 1 элемент.

int list\_queue\_add(list\_queue\_t \*list, int elem);

Описание: добавление элемента в очередь.

int list\_queue\_pop(list\_queue\_t \*list);

Описание: Удалить элемент из из очереди.

void print\_list(list\_queue\_t list);

Описание: Показать очередь.

void free\_list(list\_queue\_t \*list);

Описание: освободить память.

void print\_address\_free(size\_t \*free\_elem, int n);

Описание: вывести список освобожденных адресов.

void list\_queue\_process(void);

Описание: Смоделировать процесс обслуживания заявок с выводом состояния очередей.

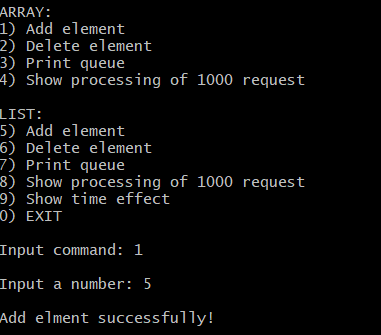
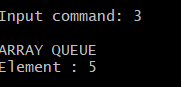
int time\_effect();

Описание: эффективность.

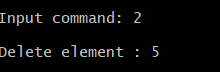
**Тесты**

а) массивом :

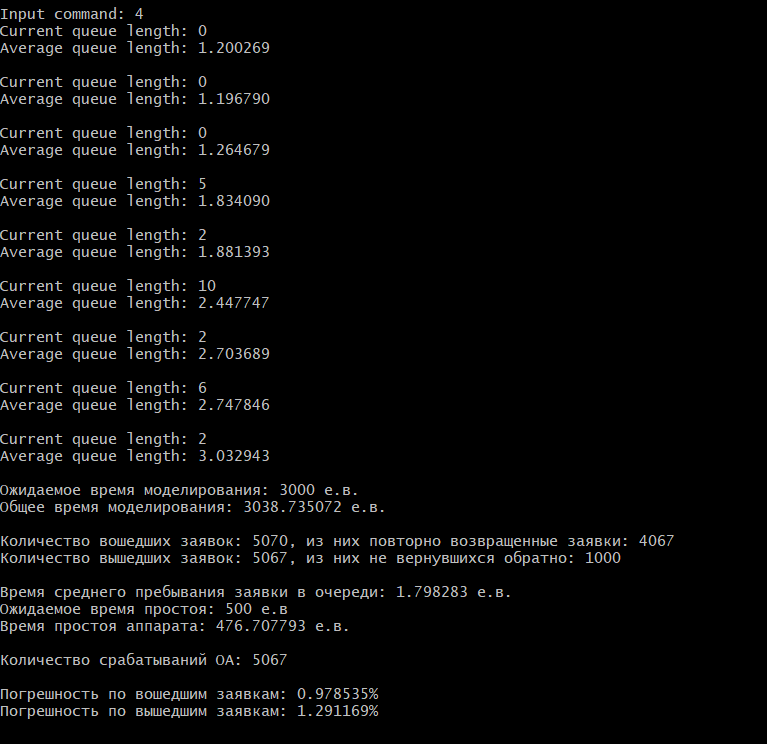
*Добавить элемент в очередь.*

*Удалить элемент из стека.*

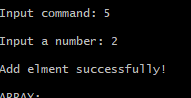
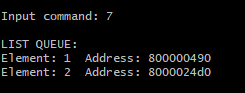
 

Смоделировать процесс обслуживания заявок с выводом состояния очередей.

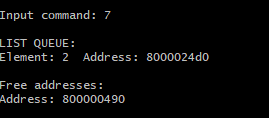


б) списком.

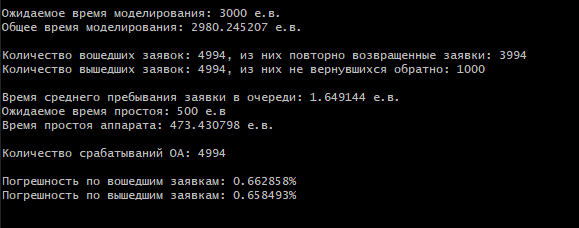
*Добавить элемент в очередь.*

*Удалить элемент из стека.*



Смоделировать процесс обслуживания заявок с выводом состояния очередей.



**Теоретический расчет времени для процесса моделирования**

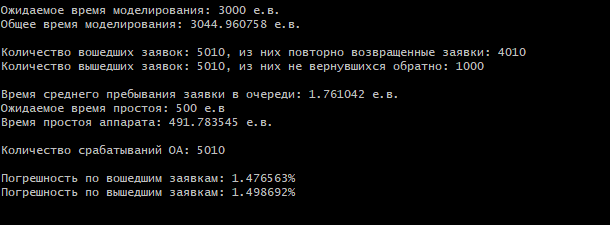
По условию задачи вероятность возврата заявки в конец очереди 0.8, а  
вероятность выхода 0.2. Тогда ОА будет работать с простоем. Время прихода -  
от 0 до 6 е.в., время обработки от 0 до 1 е.в.

Ожидаемое время моделирования будет определяться как произведение среднего интервала между приходом заявок и количеством вошедших заявок.

*real\_time* = ((0 е.в. + 6 е.в.) / 2) \* 1000 = 3000 е.в.

Ожидаемое время обработки будет определяться как произведение среднего  
времени обработки заявки, умноженного на количество обработанных заявок.  
Так как заявка выходит с вероятностью 0.2, чтобы получить 1000 заявок на  
выходе необходимо 5 проходов.  
process\_time =((0 е.в. + 1 е.в.) / 2) \* 1000 \* 5 = 2500 е.в.

Время простоя ОА определяется, как разность времени моделирования и  
времени обработки.  
wait\_time = real\_time - process\_time = 500 е.в.  
На практике при выполнении программы получены следующие резуль



Сравниваем практические результаты с теоретическими расчетами.  
Для проверки правильности работы системы по входу общее время  
моделирования делим на время прихода одной заявки:  
3044.96 / 3 = 1014.98 заявок  
Определяем предполагаемое количество вошедших заявок, фактически их  
пришло 5010 — 4010 = 1000, т.е. погрешность по вошедшим заявкам составляет  
|100%(1000 – 1014.98)/1014.98|= 1.47%  
Определим погрешность по вышедшим заявкам. Расчетное время работы равно  
3000е.в., а фактическое – 3044.96 То есть, погрешность составит  
|100%(3044.96-3000)/3000| = 1.49%.

**Оценка эффективности**

Время добавления элемента (microsecs)

|  |  |  |
| --- | --- | --- |
| Кол-во элементов | Массив | Список |
| **50** | **1** | **3** |
| **300** | **5** | **23** |
| **1000** | **8** | **118** |

Время удаления элемента из стека

|  |  |  |
| --- | --- | --- |
| Кол-во элементов | Массив | Список |
| **50** | **1** | **3** |
| **300** | **2** | **21** |
| **1000** | **8** | **84** |

Объем памяти(байты):

|  |  |  |
| --- | --- | --- |
| Кол-во элементов | Массив | Список |
| **50** | **4012** | **1248** |
| **300** | **4012** | **7248** |
| **1000** | **4012** | **24048** |

**Выводы**

Очередь, реализованная связным списком, проигрывает по времени обработки статическому массиву, так как при работе со статическим массивом необходимо работать только с указателем, а при работе со списком с указателем, а также с памятью. Но, когда заранее неизвестен максимальный размер очереди, то можно использовать связанный списки, так как в отличии от статического массива, списки ограничены в размерах только размером оперативной памяти. При реализации очереди кольцевым списком переполнение очереди возможно только в случае, когда очередь заполнена вся полностью. Фрагментация памяти при работе с очередью, реализованной при помощи односвязного списка не происходила.

Работа с массивами примерно в 10 раз быстрее, чем работа со списками, но многократно экономит память.

**Контрольные вопросы**

**Что такое очередь?**Очередь - структура данных, позволяющая добавлять элементы только в конец,а удалять только из начала. Принцип работы очереди: первым пришел – первымвышел, т.е. First In – First Out (FIFO).

**Каким образом, и какой объем памяти выделяется под хранение очереди приразличной ее реализации?**Для кольцевого статического массива: память для всей очереди выделяется настеке. Если массив динамический память выделяется на куче постепенно.Объём выделяемой памяти равен n \* sizeof (элемента очереди), где n –количество элементов в очереди.Для связного списка: память выделяется постепенно на куче при добавлениинового элемента. Объём выделяемой памяти равен n \* (sizeof(элемент очереди)+ sizeof(указатель на следующий элемент)), где n – количество элементов вочереди

**Каким образом освобождается память при удалении элемента из очереди приее различной реализации?**При хранении кольцевым массивом память не освобождается, а простоменяется указатель на конец очереди.При хранении списком, память под удаляемый кусок освобождается.

**Что происходит с элементами очереди при ее просмотре?**Элементы удаляются из очереди.

**От чего зависит эффективность физической реализации очереди?**По времени обработки выигрышной является реализация очереди в видекольцевого массива. Зная максимальный размер очереди, лучше всегоиспользовать кольцевой статический массив. Не зная максимальный размер,стоит использовать связный список, так как такую очередь можно будетпереполнить только, если закончится оперативная память.  
  
**Каковы достоинства и недостатки различных реализаций очереди взависимости от выполняемых над ней операций?**Связный список:1) недостатки: может возникнуть фрагментация памяти, а также такой способреализации является проигрышным по времени;2) достоинства: размер очереди ограничен лишь объёмом оперативной памяти.Статический кольцевой массив:1) недостатки: очередь всегда будет ограниченна определенным размером2) достоинства: операции при такой реализации выполняются гораздо быстрее.

**Что такое фрагментация памяти?**Фрагментация памяти - разбиение памяти на фрагменты, которые лежат нерядом друг с другом. Так появляются фрагменты памяти, которые нельзяиспользовать.

**На что необходимо обратить внимание при тестировании программы?**На корректное освобождение памяти и аварийные ситуации: переполнениеочереди и пустая очередь.

**Каким образом физически выделяется и освобождается память придинамических запросах?**ОС выбирает какой-то блок памяти, куда будет происходить запись данныхПри освобождении памяти ОСудаляет данные из данного блока памяти, тем самым делая его сновасвободным.