**Министерство образования и науки 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

# ЛАБОРАТОРНАЯ РАБОТА №5.

Студент Мицевич Максим Дмитриевич
*фамилия, имя, отчество*

Группа ИУ7-31Б

Студент _____ Мицевич М. Д.
*подпись, дата* *фамилия, и.о.*

Проверяющий _____ _____
*подпись, дата* *фамилия, и.о.*

Оценка _____

# 1. 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 T1, 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 T2 от 0 до 1 е.в. Каждая заявка после ОА вновь поступает в "хвост" очереди, совершая всего 5 циклов обслуживания, после чего покидает систему. (Все времена –вещественного типа) В начале процесса в системе заявок нет. Смоделировать процесс обслуживания до ухода из системы первых 1000 заявок, выдавая после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди, а в конце процесса - общее время моделирования и количестве вошедших в систему и вышедших из нее заявок, количестве срабатываний ОА, время простоя аппарата. По требованию пользователя выдать на экран адреса элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

# 2. Техническое задание

**1. Описание исходных данных**
Все входные данные читаются в виде строк и переводятся в числа. В цикле вводятся с клавиатуры числа от 0 до 11, которые определяют дальнейшие поведение программы.

Формат ввода:
Возможные пункты меню, которые можно выбрать, введя число с клавиатуры:
1 – проверка работы очереди на векторе
2 – проверка работы очереди на списке
3 – моделирование работы аппарата, используя очередь на векторе
4 - моделирование работы аппарата, используя очередь на списке
5 – сравнение эффективности обработок очереди на списке и векторе
0 – выход из подпрограммы

При добавлении заявки в очередь вводятся граничные значении времени ее входа в очередь и времени обработки. При запуске моделирования требуется ввести аналогичные данные, они будут распространяться на все заявки, которые будут учавствовать в моделировании. При создании очереди на основе вектора нужно вводить максимальное число элементов, которые могут одновременной находиться в очереди.

**2. Описание результата программы**
При моделировании работы аппарата предполагается следующий вывод на экран: после обработки каждый сотни элементов выводятся текущий и средний размер очереди, после обработки 1000 эдементов на экран выводятся общее время работы, количество вошедших в систему и вышедших из нее заявок, время простоя аппрата, а также расхождения полученных результатов с ожидаемыми.
Формат вывода:
При печати очереди в столбик печатаются ее элементы и адреса, по котоым они хранятся:
<elem_value> <elem_addres>

Ограничения:

- Время работы сортировки содержит максимум 7 значащих цифр

**3. Описание задачи, реализуемой программой**
Программа выполняет моделирование работы обслужащего аппарата с очередью, используя разные используя разные способы хранения очередей.

**4. Способ обращения к программе**

Обращение к программе происходит с помощью стандартных потоков ввода и вывода. Вызов программы app.exe

**5. Описание возможных аварийных ситуаций и ошибок пользователя**

Ошибки операционной системы при выделении динамической памяти, взятие элемента из пустой очереди, добавление в очередь, хранящуюся в виде вектора, элементов больше, чем максимальное допустимое число.

# 3. Описание внутренних структур данных

Для хранения очереди в виде вектора была выбрана данная структура:

```
typedef struct
{
   request_t *start;
   request_t *end;
   request_t *pin;
   request_t *pout;
   size_t size;
}queue_vec_t;
```

укзатели start и end — указатели на начало и конец области памяти, выделенной под очередь
pin и pout — указатели на места голову и хвост очереди

Для хранения очереди в виде списка были выбраны следующие структуры данных:

```
struct node
{
   request_t req;
   node_t *next;
};

typedef struct node node_t;
```

node_t — узел списка с элементами очереди,
req — заявка,
next — указатель на следующий элемент.

```
typedef struct
{
   node_t *head;
   node_t *tail;
}queue_list_t
```

head — указатель на начало очереди
tail - указател

# 4. Алгоритм

1. Проверяем подшло ли время добавления текущей заявки. Если да, то добавляем ее в очередь и генерируем новую текущую заявку.
2. Проверяем подошло ли время заверешения работы аппарата с текущей заявкой. Если да, то увеличиваем поле заявки с пройденным количеством итераций на 1 и изменяем состояние обслужающего аппрата. Если количество пройденных заявкой итераций равно требуемуму условием, то исключаем ее из процесса и увеличиваем счетчик вышешдий заявок на один, иначе довабляем эту заявку в конец очереди.
3. Проверяем длину очереди и состояние аппарата. Если очередь не пустая и аппарат свободен, то забираем заявку из очереди и добавляем ее в аппарат.
4. Сравниваем время до поступления текущей заявки в очередь с оставшимся временем работы аппарата. Если аппрат пустой или время до поступления заявки в очередь меньше времени работы автомата, то к общему времени работы прибавляем оставшееся до поступления заявки в очердь время, в зависимости от состояния аппарата увеличиваем время простоя или уменьшаем оставшееся время работы на ту же величину и присваиваем оставшемуся до добавления времени ноль. Иначе вычитаем из оставшегося до добавления времени и прибавляем к общему времени оставшееся время до обработки заявки, после чего присваиваем ему ноль.
5. Повторяем пункты 1-4, пока число вышедших их системы элементов не будет равно требуемуму условием.

## 5. Тесты

Позитивные тесты:

| Входные данные | Что проверяем | Ожидаемый выходной результат |
|---|---|---|
| Выбор: работы с очередью на векторе<br>Добавляем в очердь с максимальным размером 5 3 элемента 7, 8, 9 | Проверка добавления элемента в закольцованную очередь | Вектор:<br>9 - - 7 8<br>Очередь:<br>7 8 9 |
| Выбор: работа с очередью на векторе<br>Исключение из очереди, полученной в пердыдущем тесте одного элемента | Проверка удаления элемента из закольцованной очереди | Вектор:<br>- - - 7 8<br>Очередь:<br>7 8<br>Полученный элемент: 9 |
| Выбор: работа с очередью на списке<br>Добалвление в очередь элементов 1 и 2 | Проверка на добавление элементов в очередь на списке | Очередь: 1 2 |
| Выбор: работа с очередь на списке<br>Исключение элемента из очереди 1 2 3 4 | Проверка на удаление элемента из очереди на основе списка | Очередь: 2 3 4<br>Элемент: 1 |
| Выбор: моделирование работы обслужающего аппарата<br>время вхождения элемента в очередь от 0 до 1 е. в. время обслуживания от 0 до 5 е. в. элемент выходит из очереди после прохождения 5 итераций. | Проверка моделирования работы обслужащего аппрата | Общее время, деленное на 2.5, должно не больше, чем на 3% отличаться от количество вошедших элементов<br>Количество срабатываний аппрата, умноженноу на 0.5 + время простоя должно не больше, чем на 3%, отличаться от общего времени работы |

Негативные тесты:

| Входные данные | Что проверяем | Ожидаемый выходной результат |
|---|---|---|
| Выбор: Работа с очередью на списке<br>Очередь: пустая очередь | Проверка на взятие элемента из пустой очередью | Empty queue |
| Выбор: Работа с очередью на векторе<br>Очередь размером 5 элементов:<br>1 2 3 4 5 | Проверка на переполнение очереди | Queue overflow |

# 6. Функции

Добавление элемента в очередь на основе вектора и взятие из нее
**int push_qvec(void *queue, request_t req)**
**int pop_qvec(void *queue, request_t *req)**

Добавление элемента в очередь на основе списка и взятие из нее
**int push_qlist(void *queue, request_t req)**
**int pop_qlist(void *queue, request_t *req)**

Функции для создания и очистки очередей
**void free_qvec(queue_vec_t *qvec)**
**int allocate_qvec(queue_vec_t *qvec, size_t n)**
**void qlist_init(queue_list_t *qlist)**
**void free_qlist(queue_list_t qlist)**

Функция для моделирования работы обслужащего аппарата
**int model_apparat(long beg_min, long beg_max, long proc_min, long proc_max, void *queue, int (*push)(void *, request_t), int (*pop)(void *, request_t *))**

**7. Оценка эффективности программы.**

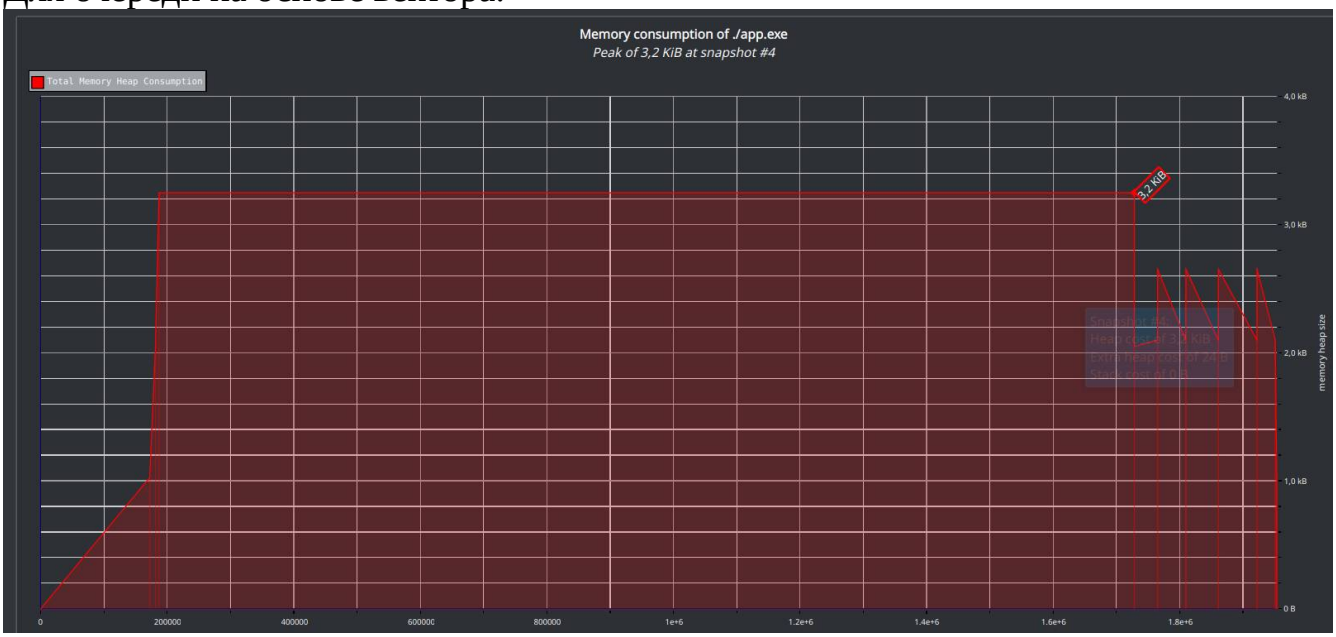
Произведем замер по времени и памяти для разного количество элементов в очереди и разного максимального размера очереди на основе вектора.

```
size - 1000
elments - 250
| list time | vector time | list memory | vector memory |
| 0.000400s |   0.000255s |       8016b |        24040b |
elments - 500
| list time | vector time | list memory | vector memory |
| 0.000771s |   0.000308s |      16016b |        24040b |
elments - 1000
| list time | vector time | list memory | vector memory |
| 0.001554s |   0.000588s |      32016b |        24040b |
size - 10000
elments - 2500
| list time | vector time | list memory | vector memory |
| 0.004069s |   0.001419s |      80016b |       240040b |
elments - 5000
| list time | vector time | list memory | vector memory |
| 0.007891s |   0.002769s |     160016b |       240040b |
elments - 10000
| list time | vector time | list memory | vector memory |
| 0.015898s |   0.005434s |     320016b |       240040b |
size - 100000
elments - 25000
| list time | vector time | list memory | vector memory |
| 0.039733s |   0.013728s |     800016b |      2400040b |
elments - 50000
| list time | vector time | list memory | vector memory |
| 0.079922s |   0.027493s |    1600016b |      2400040b |
elments - 100000
| list time | vector time | list memory | vector memory |
| 0.169869s |   0.055356s |    3200016b |      2400040b |
size - 1000000
elments - 250000
| list time | vector time | list memory | vector memory |
| 0.417098s |   0.135542s |    8000016b |     24000040b |
elments - 500000
| list time | vector time | list memory | vector memory |
| 0.854962s |   0.270552s |   16000016b |     24000040b |
```

```
elments - 1000000
| list time | vector time | list memory | vector memory |
| 1.785540s |   0.549877s |   32000016b |     24000040b |
```

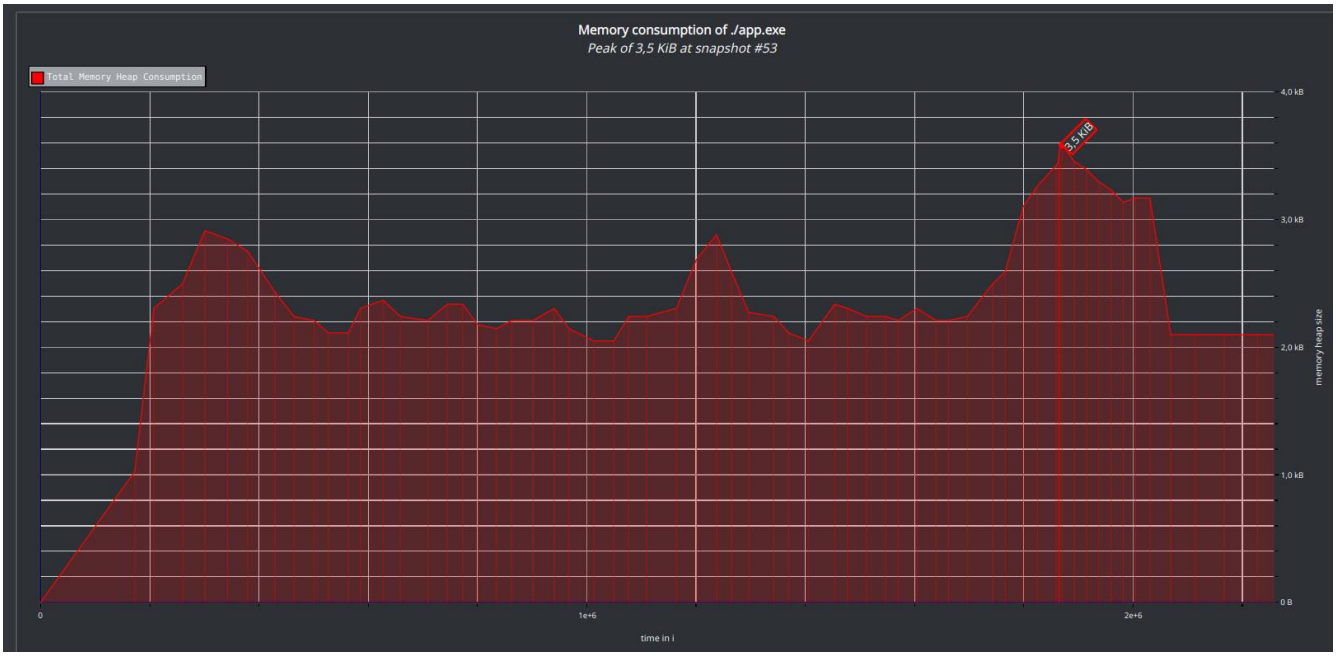
Очередь на основе списка всегда проигрывает очереди на основе вектора по времени, так как на каждую операцию push и pop в списке происходит выделение или очистка динамической памяти, в то время как в очереди на основе вектора происходит просто изменение адреса и запись элемента.
Касательно памяти, очередь на основе списка начинает выигрывать, если очередь на основе вектора заполнен меньше, чем на половину от своего максимального размера.

Еще раз удостоверимся в наших предположениях, используя анализатор памяти. Для очереди на основе вектора:




Полученный график показывает, что память для очереди на основе вектора выделяется единожды и что число элементов в очереди на основе вектора ограничена каким-то заранее определенным числом.

Для очереди на основе списка:




Данный график показывает, что память под очередь на основе списка выделяется в процессе добавления в него элементов и очищается после удаления элементов.

## 8. Выводы

Если при решении задачи основная цель — улучшение времени работы программы, то стоит использовать очередь на основе вектора. Если точное количество элементов в очереди заранее не известно, а ее размер большой, то в целях экономии памяти может быть целесообразнее использовать очередь на основе списка. Также стоит отметить, что количество элементов в очереди-списке ограничено только количеством динамической памяти, которую может выделить операционнная система под нашу программу, а не каким-то конкретным числом, как в случае очереди-вектора.

**Ответы на контрольные вопросы:**

1. Что такое очередь?
   Очередь — это последовательный список с переменной длинной, добавление элементов в который происходит с одной стороны - в хвост, а исключение элементов из которого происходит с другой строны — с головы. Очередь действует по принципу first in first out.
2. Каким образом и сколько памяти выделяется под хранение очерди при различной его реализации?
   При реализации очереди на основе вектора память под нее выделяется при ее создании. Память выделяется сплошным куском, размер выделяемой памяти равен sizeof(elem_t) * <максимальное количество элементов в очереди>
   При реализации очерди на основе списка память выделяется в процессе ее обработки, то есть при каждом вызове push создается элемент узла списка, а при каждом pop он удаляется. Размер памяти пропорционален количеству элементов в очереди — sizeof(node_t) * <количество элементов в очереди>
3. Каким образом освобождается память при удалении элемента очереди при различной его реализации?
   В очереди на основе вектора просто перемещается указатель в векторе, а память очищается, когда очередь уже не нужна.
   В очереди на основе списка память под элементом каждый раз очищается при снятии этого элемента с очереди.
4. Что происходит с элементами очереди при ее просмотре?
   Для того чтобы получить элемент, его надо снять с очереди, то есть после просмотра очереди, она становится пустой
5. Каким образом эффективнее реализовывать очередь. От чего это зависит?
   См. «Вывод»
6. В каком случае лучше реализовать очередь посредством указателей, а в каком –массивом?
   См. «Вывод»

7. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?
   Недостатки:
- при реализации на основе обычного массива часто приходится сдвигать элементы, что занимает время
- при реализации на основе колецевого массива количество элементов в очереди ограничено заранее определенным числом
- при реализации на основе списка затрачивается много времени на выделени и очистку памяти при добавлении элемента в очередь и снятии с нее
  Достоинства:
- Операции добавления и удаления в очереди на основе вектора работают относительно быстро
- Память для очереди на основе списка теоретически ограничена только размером памяти, которую может выделить операционная система

8. Что такое фрагментация памяти?
   Фрагментация — процесс появления незанятых участков в памяти. Вызвана наличием в каждом виде памяти деления на мелкие единицы фиксированного размера, в то время как объем информации необязательно кратен этому делению.
9. На что необходимо обратить внимание при тестировании программы?
   На переполнение очереди на основе вектора и на процент расхождения расчетного времени и общего времени моделирования (он должен быть в переделах 2-3 процентов)
10. Каким образом физически выделяется и освобождается память при динамических запросах?
    Выделение памяти происходит блоками — непрерывными фрагментами оперативной памяти. В какой-то момент в куче может не оказаться блока подходящего размера и, даже если свободная память достаточна для размещения объекта, операция выделения памяти окончится неудачей.