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 ПО ДИСЦИПЛИНЕ ТИПЫ И СТРУКТУРЫ ДАННЫХ

Студент **Звягин Даниил Олегович**

Группа **ИУ7-33Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Звягин Д.О.**

Преподаватель _____ **Барышникова М. Ю.**

Преподаватель _____ **Никульшина Т.А.**

Оценка _____

2023 г.

Условие задачи
Создать программу работы для работы с типом данных «очередь». Используя созданный тип данных, провести моделирование обработки запросов.

Техническое задание
Система массового обслуживания состоит из обслуживающего аппарата (ОА) и очереди заявок двух типов.
Заявки 1-го типа поступают в "хвост" очереди по случайному закону с интервалом времени T1, равномерно распределенным от 0 до 5 единиц времени (е.в.). В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время T2 от 0 до 4 е.в., после чего покидают систему.
Единственная заявка 2-го типа постоянно обращается в системе, обслуживаясь в ОА равновероятно за время T3 от 0 до 4 е.в. и возвращаясь в очередь не далее 4-й позиции от "головы".
В начале процесса заявка 2-го типа входит в ОА, оставляя пустую очередь. (Все времена – вещественного типа)
Смоделировать процесс обслуживания первых 1000 заявок 1-го типа. Выдавать после обслуживания каждых 100 заявок 1-го типа информацию о текущей и средней длине очереди, количестве вошедших и вышедших заявок и о среднем времени пребывания заявок в очереди.
В конце процесса выдать общее время моделирования, время простоя аппарата, количество вошедших в систему и вышедших из нее заявок первого типа и количество обращений заявок второго типа. По требованию пользователя выдать на экран адреса элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Входные данные:
Пользователю будет предложено меню.
Пункты меню:
0. Выход
1. Моделирование (содержимое)
2. Переключение отладки добавления элементов
3. Переключение отладки удаления элементов

Выходные данные:
Результаты моделирования, включая промежуточные данные (всё по техническому заданию)
Адреса элементов

Возможные аварийные ситуации:
Нехватка памяти

Способ обращения к программе

Собрать программу с помощью make release (/debug);
Запустить программу с помощью файла app.exe;

Структуры данных

```
// перечисление пунктов меню
enum menu_items
{
    EXIT,
    DO_TEST,
    PUSH_PRINT_SWITCH,
    POP_PRINT_SWITCH,
};

// перечисление ошибок при работе со списком
enum list_errors
{
    LIST_NO_MEMORY = 1,
    LIST_BAD_INDEX,
    LIST_NO_HEAD,
};

// тип данных узла списка
typedef struct node node_t;
struct node
{
    void *data;
    node_t *next;
};

// перечисление режимов отладки добавления и удаления элементов очереди
enum q_verbose_state
{
    Q_VERB_OFF = 0,
    Q_VERB_ON = 1,
};

// структура данных очереди, реализованной через список
typedef struct queue *queue_t;
struct queue
{
    size_t length;
    node_t *list_head;
};

// Макроопределение длины массива для очереди через массив
#define A_Q_CAPACITY 1000

// структура данных очереди, реализованной через кольцевой массив
```

```
typedef struct a_queue *a_queue_t;
struct l_queue
{
    size_t length;
    node_t *list_head;
};


// перечисление видов запросов к обслуживающему аппарату
enum req_type
{
    NO_REQUEST,
    TYPE_ONE,
    TYPE_TWO,
};

// тип данных - запрос
typedef struct request *request_t;
struct request
{
    double arrival_time;
    enum req_type type;
};

// тип данных - очередь запросов
// содержит дополнительные поля для выполнения задания
// по факту - очередь с доп. данными и функциями
// есть аналогичная структура, но для очереди через массив
typedef struct req_queue *req_queue_t;
struct req_queue
{
    double time_now;
    double sleep_time;
    double wait_sum;
    double length_sum;
    size_t t1_in;
    size_t t1_out;
    size_t t2_in;
    size_t t2_out;
    queue_t queue;
};
```

Теоретический расчёт

Поскольку в моём варианте работы время обработки элемента очереди меньше времени прихода очередного запроса, очередь (в среднем) не должна увеличиваться, а общее время симуляции должно вычисляться, как кол-во элементов * среднее время прихода запроса, то есть 1000*2.5=2500

При запуске симуляций я наблюдаю значения (в среднем) около 2550-2600 единиц времени для обоих типов очередей, то есть значение больше, чем вычисленное математически, однако при вычислении этого значения я не брал в учёт циркулирующий в системе запрос типа 2, который несколько смещает ожидаемое время.

Пример работы

```
Меню:
0: Выход
1: Провести симуляцию
2: Переключить отладку добавки     элементов (сейчас - выкл)
3: Переключить отладку уничтожения элементов (сейчас - выкл)
> 1
Обработано 100 заявок первого типа
Текущая длина очереди: 4
Средняя длина очереди: 4.54
Среднее время в очереди: 31.70
Всего заявок обработано: 134
Только за последний промежуток: 134

Обработано 200 заявок первого типа
Текущая длина очереди: 7
Средняя длина очереди: 4.48
Среднее время в очереди: 51.97
Всего заявок обработано: 262
Только за последний промежуток: 128

Обработано 300 заявок первого типа
Текущая длина очереди: 4
Средняя длина очереди: 5.52
Среднее время в очереди: 74.10
Всего заявок обработано: 388
Только за последний промежуток: 126

Обработано 400 заявок первого типа
Текущая длина очереди: 4
Средняя длина очереди: 5.13
Среднее время в очереди: 101.22
Всего заявок обработано: 520
Только за последний промежуток: 132

Обработано 500 заявок первого типа
Текущая длина очереди: 3
Средняя длина очереди: 4.59
Среднее время в очереди: 139.99
Всего заявок обработано: 664
Только за последний промежуток: 144

Обработано 600 заявок первого типа
Текущая длина очереди: 9
Средняя длина очереди: 5.23
Среднее время в очереди: 157.27
Всего заявок обработано: 791
Только за последний промежуток: 127
```

```
Обработано 700 заявок первого типа
Текущая длина очереди: 11
Средняя длина очереди: 5.53
Среднее время в очереди: 173.33
Всего заявок обработано: 916
Только за последний промежуток: 125

Обработано 800 заявок первого типа
Текущая длина очереди: 6
Средняя длина очереди: 5.69
Среднее время в очереди: 190.21
Всего заявок обработано: 1041
Только за последний промежуток: 125

Обработано 900 заявок первого типа
Текущая длина очереди: 1
Средняя длина очереди: 5.40
Среднее время в очереди: 220.59
Всего заявок обработано: 1177
Только за последний промежуток: 136

Обработано 1000 заявок первого типа
Текущая длина очереди: 1
Средняя длина очереди: 5.34
Среднее время в очереди: 244.83
Всего заявок обработано: 1309
Только за последний промежуток: 132

Обработка окончена
Общее время моделирования: 2602.843575
Время простоя Аппарата: 1.375644
Вошло и вышло заявок первого типа: 1000
Вошло и вышло заявок второго типа: 309
```

Теоретический расчёт 2

Если изменить время прихода и обработки элементов на обратные (до 4 приход, до 5 - обработка), время симуляции сместится, опять же из-за дополнительной циркуляции запроса второго типа
и мы получим 1000*2.5 = 2500 плюс 250 (в среднем - запросов второго типа) * 2.5 = 625
в сумме - 3125 единиц времени. В симуляции получил 3080 единиц - следовательно модель работает. Скриншоты приложу ниже

```
Обработано 500 заявок первого типа
Текущая длина очереди: 281
Средняя длина очереди: 133.52
Среднее время в очереди: 334.47
Всего заявок обработано: 626
Только за последний промежуток: 125

Обработано 600 заявок первого типа
Текущая длина очереди: 332
Средняя длина очереди: 161.61
Среднее время в очереди: 404.32
Всего заявок обработано: 751
Только за последний промежуток: 125

Обработано 700 заявок первого типа
Текущая длина очереди: 301
Средняя длина очереди: 186.51
Среднее время в очереди: 474.21
Всего заявок обработано: 876
Только за последний промежуток: 125

Обработано 800 заявок первого типа
Текущая длина очереди: 201
Средняя длина очереди: 194.54
Среднее время в очереди: 542.86
Всего заявок обработано: 1001
Только за последний промежуток: 125

Обработано 900 заявок первого типа
Текущая длина очереди: 101
Средняя длина очереди: 189.68
Среднее время в очереди: 607.74
Всего заявок обработано: 1126
Только за последний промежуток: 125

Обработано 1000 заявок первого типа
Текущая длина очереди: 1
Средняя длина очереди: 175.80
Среднее время в очереди: 673.70
Всего заявок обработано: 1251
Только за последний промежуток: 125

Обработка окончена
Общее время моделирования: 3078.243495
Время простоя Аппарата: 2.201035
Вошло и вышло заявок первого типа: 1000
Вошло и вышло заявок второго типа: 251
Обработка заявок очередью через массив
```

Замеры

В данной лабораторной работе не имеют смысла замеры скорости или памяти, однако можно узнать погрешность и примерно определить правильность работы программы, если вычислить примерное время работы математически.

Я считаю, что 2600 ед. времени - достаточно близкий к ожидаемому (даже без второго запроса) результат.

Что касается проведения самой симуляции, симуляция очереди через список занимает в среднем 22000 нс, а через массив - 6000 нс;

Выводы по проделанной работе

Выводы о фрагментации
На моей машине фрагментация памяти не наблюдается (см. скриншот ниже). На ней видно, что разница между всеми указателями постоянна

```
queue push pointer        0x559b85ad55b0
queue push pointer        0x559b85ad55f0
queue push pointer        0x559b85ad5630
queue push pointer        0x559b85ad5670
queue push pointer        0x559b85ad56b0
queue push pointer        0x559b85ad56f0
queue push pointer        0x559b85ad5730
queue push pointer        0x559b85ad5770
queue push pointer        0x559b85ad57b0
queue push pointer        0x559b85ad57f0
queue push pointer        0x559b85ad5830
queue push pointer        0x559b85ad5870
queue push pointer        0x559b85ad58b0
queue push pointer        0x559b85ad58f0
queue push pointer        0x559b85ad5930
queue push pointer        0x559b85ad5970
queue push pointer        0x559b85ad59b0
queue push pointer        0x559b85ad59f0
queue push pointer        0x559b85ad5a30
queue push pointer        0x559b85ad5a70
queue push pointer        0x559b85ad5ab0
queue push pointer        0x559b85ad5af0
queue push pointer        0x559b85ad5b30
queue push pointer        0x559b85ad5b70
queue push pointer        0x559b85ad5bb0
queue push pointer        0x559b85ad5bf0
queue push pointer        0x559b85ad5c30
queue push pointer        0x559b85ad5c70
queue push pointer        0x559b85ad5cb0
queue push pointer        0x559b85ad5cf0
queue push pointer        0x559b85ad5d30
queue push pointer        0x559b85ad5d70
queue push pointer        0x559b85ad5db0
queue push pointer        0x559b85ad5df0
queue push pointer        0x559b85ad5e30
queue push pointer        0x559b85ad5e70
queue push pointer        0x559b85ad5eb0
queue push pointer        0x559b85ad5ef0
queue push pointer        0x559b85ad5f30
queue push pointer        0x559b85ad5f70
queue push pointer        0x559b85ad5fb0
queue push pointer        0x559b85ad5ff0
queue push pointer        0x559b85ad6030
queue push pointer        0x559b85ad6070
queue push pointer        0x559b85ad60b0
queue push pointer        0x559b85ad60f0
queue push pointer        0x559b85ad6130
queue push pointer        0x559b85ad6170
queue push pointer        0x559b85ad61b0
```

Очередь, реализованная с помощью односвязного списка, стабильно проигрывает очереди, выполненной в виде статического массива по скорости от 3 до 5 раз, однако при неопределённом количестве запросов за единицу времени, очередь, реализованная с помощью списка гарантирует стабильность и правильность работы (отсутствие переполнения), что невозможно гарантировать при исполнении через массив, а также занимает только столько памяти, сколько нужно в данный момент (хоть на каждый элемент и приходится выделять дополнительную память для хранения адреса следующего узла). Таким образом, можно сделать вывод, что если нужно реализовать такую структуру данных как очередь, то нужно, ориентируясь на объём доступной памяти и ожидаемое количество одновременных запросов выбрать либо очередь в виде массива – для прироста в скорости. Однако если вы рассчитываете обрабатывать меньшее количество элементов и вам критически важен объём занимаемой памяти, или вам не важна память, но количество запросов может быть очень большим, то можно выбрать список для обеспечения стабильности работы программы.

Контрольные вопросы

1. Что такое FIFO и LIFO?
FIFO — FIRST IN FIRST OUT - принцип работы как у очереди. На выход сначала поступают те элементы, которые пришли раньше всего
LIFO — LAST IN FIRST OUT - принцип работы как у стека. На выход сначала поступают те элементы, которые пришли позже всего

2. Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?
При реализации массивом объем памяти рассчитывается путем перемножения количества элементов на размер одного элемента, то есть то количество памяти, которое выделено на массив - оно не меняется. Если массив статический, то память выделяется в стеке, если массив динамический, то в куче.
При реализации списком, под каждый новый элемент выделяется память размером (размер элемента + размер указателя на следующий узел списка), для каждого элемента отдельно. Память выделяется только тогда, когда она нужна, поэтому размер такой реализации со временем меняется.

3. Каким образом освобождается память при удалении элемента из очереди при различной её реализации?
При удалении элемента из очереди в виде массива, перемещается указатель чтения, сама память в этот момент не освобождается. Память всего

массива, если он статический, освобождается после того, как закончится область его видимости. Если массив динамический, память после работы с массивом необходимо освободить с помощью функции free().
При удалении элемента из очереди в виде списка, память, выделенная под данный элемент освобождается сразу. (Указатель на «голову» переходит на следующий элемент, считанный элемент удаляется, память освобождается).

4. Что происходит с элементами очереди при ее просмотре?
При просмотре очереди, головной элемент («голова») удаляется (Или для массива - перемещается указатель чтения и память считается удалённой, хоть и не является таковой), и указатель смещается. То есть при просмотре очереди ее элементы удаляются.

5. От чего зависит эффективность физической реализации очереди?
При реализации очереди в виде циклического массива, может возникнуть переполнение очереди, если указатель записи "догонит" указатель чтения, однако по определению массива фрагментация памяти невозможна. Быстрее работают операции добавления и удаления элементов. Также необходимо знать тип данных элемента массива.
При реализации обычным массивом, вставка и удаление элементов будет происходить дольше, что сильно уменьшит преиущества выполнения очереди в виде массива.
При реализации в виде списка легче удалять и добавлять элементы, переполнение памяти может возникнуть только если закончится память на куче, однако может возникнуть фрагментация памяти.
Если изначально знать размер очереди и тип данных, то лучше воспользоваться массивом. Не зная размер — списком.
Если важна скорость выполнения, то лучше использовать массив, так как все операции с массивом выполняются быстрее, однако возникает вопрос стабильности при возможном переполнении.

6. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?
При реализации очереди в виде массива не возникает фрагментация памяти, однако может возникнуть переполнение очереди, и тратится дополнительное время на сдвиги элементов (например, как в моём варианте, вставка не далее 4-й позиции).
При реализации очереди в виде списка, проще выполнять операции добавления и удаления элементов, но может возникнуть фрагментация памяти.

7. Что такое фрагментация памяти, и в какой части ОП она возникает?
Фрагментация – чередование участков памяти при последовательных запросах на выделение и освобождение памяти. «Занятые» участки чередуются со «свободными» - однако последние могут быть недостаточно большими для того, чтобы сохранить в них нужное данное.

8. Для чего нужен алгоритм «близнецов»?
Скорость выделения памяти можно существенно повысить, если выделять ее блоками заранее определенного размера, при этом может быть одновременно использовано несколько альтернативных размеров блока. Скорость выделения памяти возрастает потому, что не требуется искать в списке подходящий по размеру блок.
Метод близнецов обеспечивает очень высокую скорость динамического выделения и освобождения памяти, и используется в составе многих современных операционных систем для динамического распределения памяти в ядре системы, драйверах или в других ответственных компонентах системы, критичных к скорости работы.

9. Какие дисциплины выделения памяти вы знаете?
Две основные дисциплины сводятся к принципам "самый подходящий" и "первый подходящий". По дисциплине "самый подходящий" выделяется тот свободный участок, размер которого равен запрошенному или превышает его на минимальную величину. По дисциплине "первый подходящий" выделяется первый же найденный свободный участок, размер которого не меньше запрошенного. При применении любой дисциплины, если размер выбранного для выделения участка превышает запрос, выделяется запрошенный объем памяти, а остаток образует свободный блок меньшего размера.

10. На что необходимо обратить внимание при тестировании программы?
При реализации очереди в виде списка необходимо следить за освобождением памяти при удалении элемента из очереди. Если новые элементы приходят быстрее, чем уходят старые, то может возникнуть фрагментация памяти. При реализации очереди в виде массива надо обратить внимание на отсутствие переполнения, а также проверять выход за пределы выделенной памяти (однако упустить ошибку сегментации довольно трудно).

11. Каким образом физически выделяется и освобождается память при динамических запросах?
Программа дает запрос ОС на выделение блока памяти необходимого размера. ОС находит подходящий блок, записывает его адрес и размер в таблицу адресов, а затем возвращает данный адрес в программу.
При запросе на освобождение указанного блока программы, ОС убирает его из таблицы адресов, однако указатель на этот блок может остаться в программе. Обращение к этому адресу и попытка считать данные из этого блока может привести к неопределенному поведению, так как данные могут быть уже изменены.