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

Обработка очередей

Вариант 7

Студент **Ильченко Е. А.**

Группа **ИУ7-34Б**

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

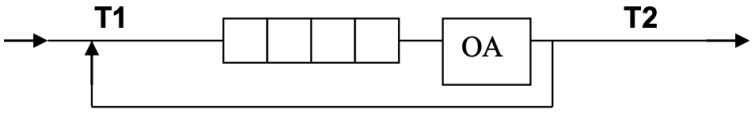
Студент _____ **Ильченко Е. А.**

Преподаватель _____ **Силантьева А. В.**

2024 г.

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 адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Описание ТЗ

1. Описание исходных данных и результатов работы программы

Входные данные:
Пользовательская команда из доступных и необходимые аргументы определенного сценария:

1: Добавить элементы в очередь на массиве
2: Удалить элементы из очереди на массиве
3: Вывести адреса элементов очереди на массиве
4: Добавить элементы в очередь на списке
5: Удалить элементы из очереди на списке
6: Вывести адреса элементов очереди на списке
7: Запустить моделирование
8: Вывести меню
9: Замер времени
0: Выход

Для моделирования: интервал времени поступления заявки, интервал времени обработки заявки, количество циклов обслуживания

Выходные данные:
При работе с очередью: очередь, измененная в соответствии с выбранной операцией
При моделировании: общее время моделирования, погрешность, время простоя аппарата, количество вошедших и вышедших заявок, средняя длина очереди, количество срабатываний ОА

2. Описание задачи, реализуемой в программе

Задача, реализуемая в программе, заключается в реализации операций для работы с очередью на массиве и на списке: добавление элемента в очередь, удаление элемента из очереди, вывод адресов элементов очереди; сравнении времени работы алгоритмов добавления элементов в очередь и удаления элементов из нее; моделировании задания по варианту.

3. Способ обращения к программе

Запуск исполняемого файла

```
./app.exe
```

Далее выбирается, какой пункт меню выполнить

4. Описание возможных аварийных ситуаций и ошибок пользователя

- Команда введена неверно
- Количество элементов для добавления введено неверно
- Количество элементов для удаления введено неверно
- Удаление из пустого стека
- Вставка в переполненный стек
- Неверный ввод данных для моделирования

5. Описание внутренних структур данных

Запрос

```
typedef struct
{
   int cycles_left;
   double arrival_time;
   double processing_time;
} Request;
```

Очередь на массиве

```
typedef struct
{
   Request data[LIST_CAPACITY];
   int head;
   int tail;
   int size;
} ArrayQueue;
```

Очередь на списке

```
typedef struct Node
{
   Request data;
   struct Node *next;
} Node;

typedef struct
{
   Node *tail;
   Node *head;
   int size;
} ListQueue;
```

6. Описание функций

Функции для работы с очередью на массиве

```
int push_queue_arr(ArrayQueue *queue, Request request)
```

Добавление элемента в очередь

```
int push_many_queue_arr(ArrayQueue *queue, Request request, int n);
```

Добавление n элементов в очередь

```
int pop_queue_arr(ArrayQueue *queue, Request *request)
```

Удаление элемента из очереди

```
int pop_many_queue_arr(ArrayQueue *queue, Request *request, int n);
```

Удаление n элементов из очереди

```
int print_queue_arr(ArrayQueue *queue)
```

Вывод очереди

Функции для работы с очередь на списке

```
void init_queue_list(ListQueue *queue)
```
Инициализация очереди

```
int push_queue_list(ListQueue *queue, Request request)
```
Добавление элемента в очередь

```
int push_many_queue_list(ListQueue *queue, Request request, int n);
```
Добавление n элементов в очередь

```
int pop_queue_list(ListQueue *queue, Request *request)
```
Удаление элемента из очереди

```
int pop_many_queue_list(ListQueue *queue, Request *request, int n);
```
Удаление n элементов из очереди

```
int print_queue_list(ListQueue *queue)
```
Вывод очереди

```
void free_queue_list(ListQueue *queue)
```
Освобождение очереди

Моделирование и эффективность

```
void modeling(int t1_start, int t1_end, int t2_start, int t2_end, int cycles)
```
Моделирование процесса по варианту

```
void efficiency(void);
```
Замер эффективности добавления и удаления элементов из очереди на массиве и на списке

7. Описание алгоритма

Алгоритм добавления элемента в очереди на массиве

1. Инициализация данных: Передается указатель на очередь `queue` и добавляемый элемент `request`.
2. Проверка заполненности очереди: Вызывается функция `is_full_arr(queue)`. Если возвращается истина (очередь заполнена), операция завершается, возвращая 0.
3. Добавление элемента в очередь: Элемент `request` записывается в массив `queue->data` по индексу `queue->tail`.

4. Обновление индекса хвоста: Индекс `queue->tail` обновляется с учетом циклического перемещения: `(queue->tail + 1) % LIST_CAPACITY`.
5. Увеличение размера очереди: Поле `queue->size` увеличивается на единицу.
6. Завершение операции: Возвращается 1, что указывает на успешное добавление.

Алгоритм удаления элемента из очереди на массиве

1. Инициализация данных: Передается указатель на очередь `queue` и указатель на переменную `request`, куда будет сохранен удаляемый элемент.
2. Проверка пустоты очереди: Вызывается функция `is_empty_arr(queue)`. Если возвращается истина (очередь пуста), операция завершается, возвращая 0.
3. Извлечение элемента из очереди: Элемент из массива `queue->data` по индексу `queue->head` сохраняется в переменную, на которую указывает `request`.
4. Обновление индекса головы: Индекс `queue->head` обновляется с учетом циклического перемещения: `(queue->head + 1) % LIST_CAPACITY`.
5. Уменьшение размера очереди: Поле `queue->size` уменьшается на единицу.
6. Завершение операции: Возвращается 1, что указывает на успешное удаление.

Алгоритм добавления элемента в очередь на списке

1. Инициализация данных: Передается указатель на очередь `queue` и добавляемый элемент `request`.
2. Создание нового узла: Выделяется память под узел с помощью `malloc(sizeof(Node))`. Если память не была выделена, операция завершается с ошибкой, возвращая 0.
3. Заполнение узла: Поле `node->data` заполняется значением `request`. Поле `node->next` устанавливается в `NULL`, так как новый узел будет находиться в хвосте очереди.
4. Добавление узла в очередь:
    a. Если очередь пуста (`is_empty_list(queue)` возвращает истину): Поля `queue->head` и `queue->tail` указываются на созданный узел.
    b. Если очередь не пуста: Поле `queue->tail->next` указывает на новый узел, а `queue->tail` обновляется, чтобы указывать на этот узел.
5. Обновление размера очереди: Поле `queue->size` увеличивается на единицу.
6. Завершение операции: Возвращается 1, что указывает на успешное добавление элемента.

Алгоритм удаления элемента из очереди на списке

1. Инициализация данных: Передается указатель на очередь `queue` и указатель на переменную `request`, куда будет сохранен удаляемый элемент.

2. Проверка пустоты очереди: Вызывается функция `is_empty_list(queue)`. Если возвращается истина (очередь пуста), операция завершается, возвращая 0.
3. Извлечение элемента из головы очереди: Указатель `tmp` устанавливается на текущую голову очереди (`queue->head`). Значение узла `tmp->data` копируется в переменную, на которую указывает `request`.
4. Обновление головы очереди: Поле `queue->head` сдвигается к следующему элементу (`queue->head->next`). Если новый `queue->head` равен `NULL`, также обновляется `queue->tail` (устанавливается в `NULL`).
5. Освобождение памяти: Вызывается `free(tmp)` для удаления старой головы очереди.
6. Обновление размера очереди: Поле `queue->size` уменьшается на единицу.
7. Завершение операции: Возвращается 1, что указывает на успешное удаление элемента.

Алгоритм моделирования

1. Инициализация данных:
    a. Инициализируется очередь в виде массива (`ArrayQueue`) и списка (`ListQueue`). Для списка вызывается `init_queue_list`.
    b. Устанавливается текущее время `current_time` в 0.
    c. Генерируются моменты времени следующего прибытия заявки (`next_arrival`) и следующего завершения обработки (`next_departure`).
2. Инициализация параметров статистики:
    a. Общее число вошедших заявок (`total_arrivals`) и вышедших заявок (`total_departures`) устанавливаются в 0.
    b. Переменные для фиксации времени простоя аппарата (`idle_time`) и подсчёта средней длины очереди (`average_length_sum`) инициализируются.
3. Основной цикл моделирования:
   Выполняется до тех пор, пока число обработанных заявок (`total_departures`) не достигнет заданного значения (`TOTAL_PROCESSED`).
4. Обработка поступления новой заявки:
    a. Если текущее время `current_time` больше или равно `next_arrival`:
        i. Создаётся новая заявка с фиксированным количеством циклов обработки (`cycles_left`) и временем прибытия (`arrival_time`).
        ii. Генерируется случайное время обработки заявки (`processing_time`).
        iii. Заявка добавляется в обе очереди с помощью `push_queue_arr` и `push_queue_list`. Если добавление неуспешно, выводится сообщение об ошибке.
        iv. Обновляется время следующего прибытия заявки (`next_arrival`).

5. Обработка завершения обслуживания заявки:
    a. Если аппарат занят (`is_processing`) и текущее время `current_time` больше или равно `next_departure`:
        i. Уменьшается количество оставшихся циклов обработки заявки (`cycles_left`).
        ii. Если заявка требует повторной обработки (циклы не завершены), она возвращается в обе очереди. При ошибке добавления выводится сообщение.
        iii. Если циклы завершены, увеличивается счётчик вышедших заявок (`total_departures`).
        iv. Аппарат освобождается (`is_processing = 0`).
6. Запуск обработки следующей заявки:
    a. Если аппарат свободен (`is_processing == 0`) и очередь массива содержит элементы (`array_queue.size > 0`):
        i. Извлекается заявка из обеих очередей с помощью `pop_queue_arr` и `pop_queue_list`.
        ii. Устанавливается время завершения обработки (`next_departure`), и аппарат помечается как занятый (`is_processing = 1`).
7. Учёт простоя аппарата:
   Если очередь пуста (`array_queue.size == 0`) и аппарат свободен, время простоя (`idle_time`) увеличивается на фиксированный шаг времени (`TIME_STEP`).
8. Подсчёт средней длины очереди:
   Величина очереди массива (`array_queue.size`) добавляется к сумме для подсчёта средней длины (`average_length_sum`).
9. Обновление текущего времени:
   Текущее время увеличивается на шаг (`current_time += TIME_STEP`).
10. Периодический вывод статистики:
    a. Если количество обработанных заявок (`total_departures`) кратно 100 и новое значение превышает ранее зафиксированное (`last_reported`):
    b. Обновляется переменная `last_reported`.
        i. Выводится информация о текущей длине очереди для массива и списка, а также о средней длине очереди.
11. Освобождение ресурсов:
    Освобождается память, выделенная для списка (`free_queue_list`).
12. Вывод итоговых результатов:
    Отображаются общее время моделирования, время простоя аппарата, количество вошедших и вышедших заявок, а также средняя длина очереди.

Тесты

Тест	Входные данные	Выходные данные
Некорректный ввод пункта меню	100	Неверный ввод команды
Переполнение очереди	Количество элементов больше, чем размер массива	Стек заполнен
Попытка удалить элементы из пустой очереди	Пустая очередь и команда удаления элемента	Стек пуст
Неверный ввод количества элементов для добавления	При запросе ввода количества элементов вводится: a	Неверно введено количество элементов для добавления
Неверный ввод количества элементов для удаления	При запросе ввода количества элементов вводится: a	Неверно введено количество элементов для удаления
Неверный ввод интервала времени поступления заявок	Введите границы интервала: 0 a	Ошибка ввода
Неверный ввод интервала времени обработки заявок	Введите границы интервала: 0 a	Ошибка ввода
Неверный ввод количества циклов обслуживания	Введите количество циклов: -1	Ошибка ввода

Оценка эффективности

1. Операции вставки и удаления для очереди на массиве и на списке

Эффективность по времени/памяти считалась путем замера 1000 раз методов удаления и добавления элементов и усреднения результатов
Выгода считалась, как

$$\frac{List - Array}{Array} \cdot 100$$

Измерения проводились на MacBook Pro 13 2019

Добавление/удаление **10** элементов

Очередь как массив			**Очередь как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка	Удаление	Память
219.08	183.81	240016	550.63	526.71	240	151.33	186.55	-99.90

Добавление/удаление **110** элементов

Очередь как массив			**Очередь как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка	Удаление	Память
1555.50	1484.37	240016	5057.57	5607.87	2640	225.14	277.79	-98.90

Добавление/удаление **210** элементов

Очередь как массив			**Очередь как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка	Удаление	Память
2874.69	2755.85	240016	8664.30	10123.45	5040	201.40	267.34	-97.90

Добавление/удаление **310** элементов

Очередь как массив			**Очередь как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка	Удаление	Память
4163.40	4038.98	240016	12400.25	14479.13	7440	197.84	258.48	-96.90

Добавление/удаление **410** элементов

Очередь как массив			**Очередь как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Вставка, сек$^{-9}$	Удаление, сек$^{-9}$	Память, байт	Вставка, сек$^{-9}$	Удаление, сек$^{-9}$	Память, байт	Вставка	Удаление	Память
5721.79	6343.82	240016	18164.46	19739.13	9840	217.46	211.16	-95.90

Добавление/удаление **510** элементов

Очередь как массив			**Очередь как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Вставка, сек$^{-9}$	Удаление, сек$^{-9}$	Память, байт	Вставка, сек$^{-9}$	Удаление, сек$^{-9}$	Память, байт	Вставка	Удаление	Память
7953.46	6728.35	240016	19846.49	22947.63	12240	149.53	241.06	-94.90

Добавление/удаление **610** элементов

Очередь как массив			**Очередь как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Вставка, сек$^{-9}$	Удаление, сек$^{-9}$	Память, байт	Вставка, сек$^{-9}$	Удаление, сек$^{-9}$	Память, байт	Вставка	Удаление	Память
8258.07	8346.30	240016	24672.95	33750.53	14640	198.77	304.38	-93.90

Добавление/удаление **710** элементов

Очередь как массив			**Очередь как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Вставка, сек$^{-9}$	Удаление, сек$^{-9}$	Память, байт	Вставка, сек$^{-9}$	Удаление, сек$^{-9}$	Память, байт	Вставка	Удаление	Память
9991.14	10498.17	240016	31678.85	36848.89	17040	217.07	251.00	-92.90

Добавление/удаление **810** элементов

Очередь как массив			**Очередь как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка	Удаление	Память
39669.15	11278.58	240016	50366.02	41470.84	19440	26.97	267.70	-91.90

Добавление/удаление **910** элементов

Очередь как массив			**Очередь как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка	Удаление	Память
12007.57	12045.00	240016	40994.28	46244.20	21840	241.40	283.93	-90.90

Добавление/удаление **1010** элементов

Очередь как массив			**Очередь как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка, $сек^{-9}$	Удаление, $сек^{-9}$	Память, байт	Вставка	Удаление	Память
13776.79	13462.22	240016	41188.83	51292.57	24240	198.97	281.01	-89.90

2. Оценка моделирования

<u>Время моделирования определяется временем прихода</u>

Результаты моделирования

```
Общее время моделирования: 3068.90 е.в.
Погрешность: 2.3 %
Время простоя аппарата: 277.90 е.в.
Всего вошло заявок: 1005
Всего вышло заявок: 1000
Средняя длина очереди: 3.38
Количество срабатываний обслуживающего аппарата: 5010
```

Среднее время прихода одной заявки: (0 + 6) / 2 = 3 е.в.

Среднее время обработки одной заявки: (0 + 1) / 2 = 0.5 е.в.

Расчетное время работы: 3 * 1000 = 3000 е.в.

Проверка правильности работы системы по входу
Предполагаемое количество вошедших заявок: 3068.90 / 3 = 1022.97
Погрешность: 100 * (1005 - 1022.97) / 1022.97 = 1.75 % – в пределах допустимого

Проверка правильности работы системы по выходу
Погрешность: 100 * (3068.90 - 3000) / 3000 = 2.3 % – в пределах допустимого

Время моделирования определяется временем обработки
Интервал времени обработки заявки [0; 2]
Результаты моделирования

```
Общее время моделирования: 7636.10 е.в.
Погрешность: 2.88 %
Время простоя аппарата: 4.10 е.в.
Всего вошло заявок: 2507
Всего вышло заявок: 1000
Средняя длина очереди: 726.21
Количество срабатываний обслуживающего аппарата: 7422
```

Среднее время прихода одной заявки: (0 + 6) / 2 = 3 е.в.
Среднее время обработки одной заявки: (0 + 2) / 2 = 1 е.в.

Проверка правильности работы системы по входу
Предполагаемое количество вошедших заявок: 7636.10 / 3 = 2545.37
Погрешность: 100 * (2545.37 - 2507) / 2545.37 = 1.51 % – в пределах допустимого

Проверка правильности работы системы по выходу
Погрешность: 100 * (7636.10 - 7422) / 7422 = 2.88 % – в пределах допустимого

Массив

```
Общее время моделирования: 3001.70 е.в.
Погрешность: 0.06 %
Время простоя аппарата: 250.20 е.в.
Всего вошло заявок: 1000
Всего вышло заявок: 1000
Средняя длина очереди: 3.31
Количество срабатываний обслуживающего аппарата: 5000
```

Список

```
Результаты моделирования:
Общее время моделирования: 3101.30 е.в.
Погрешность: 3.38 %
Время простоя аппарата: 378.00 е.в.
Всего вошло заявок: 1004
Всего вышло заявок: 1000
Средняя длина очереди: 2.63
Количество срабатываний обслуживающего аппарата: 5008
```

Вывод

Моделирование работы системы массового обслуживания:
В результате моделирования процесса обслуживания заявок с использованием очередей на основе массива и списка были достигнуты поставленные цели: реализован процесс обработки первых 1000 заявок с анализом их времени пребывания в системе, длины очередей и времени простоя обслуживающего аппарата.

Сравнение реализации очередей:

1. **Очередь на массиве**:
    a. Преимущества: Высокая скорость доступа к элементам благодаря прямому индексации. Простая реализация и управление (меньше накладных расходов на управление памятью).
    b. Недостатки: Фиксированный размер массива, что может приводить к переполнению или избыточному использованию памяти. Ограничения при динамическом изменении размера очереди.
2. **Очередь на списке**:
    a. Преимущества: Возможность динамического изменения размера очереди. Экономия памяти при большом количестве операций, если элементы удаляются из начала очереди.
    b. Недостатки: Более сложная реализация, включая управление выделением и освобождением памяти. Более высокая нагрузка на память из-за хранения указателей и дополнительных затрат на динамическое выделение памяти.

Эффективность реализации:

1. По времени выполнения:

    a. Очередь на массиве продемонстрировала более быстрое выполнение операций добавления и удаления за счёт отсутствия операций выделения/освобождения памяти по сравнению с очередью на списке
2. По использованию памяти:
    a. Очередь на списке использует память более экономно при небольших объёмах данных, однако при увеличении размера очереди возникают накладные расходы на хранение указателей.
    b. Очередь на массиве показала большую эффективность при постоянной длине очереди, но при увеличении объёма данных фиксированный размер массива создаёт ограничения: невозможность выделить цельный участок памяти для большого количества элементов из-за ограниченного объема памяти устройства и фрагментации памяти

Анализ работы обслуживающего аппарата:

1. Время простоя аппарата оказалось минимальным, так как среднее время обработки заявок меньше среднего интервала их поступления.
2. Время моделирования и количество заявок соответствовали теоретически ожидаемым значениям с допустимым отклонением в пределах 2-3%.

Особенности работы с памятью:

При реализации очереди на списке были зафиксированы адреса выделения и освобождения памяти. Это позволило наблюдать фрагментацию памяти, которая может возникать при частых операциях добавления и удаления элементов.

Ответы на контрольные вопросы

1. **Что такое FIFO и LIFO?**
   FIFO (First-In-First-Out): Это метод управления данными, при котором элемент, добавленный первым, обрабатывается первым, а последний добавленный элемент обрабатывается последним. Данный метод используется в структуре данных стек

   LIFO (Last-In-First-Out): В этом методе последний добавленный элемент обрабатывается первым, а первый добавленный элемент обрабатывается последним. Данный метод используется в структуре данных очередь
2. **Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?**
   В случае реализации с помощью списка память выделяется динамически под каждый узел очереди. В случае реализации статическим массивом память выделяется на стеке единым куском под все элементы.

3. **Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?**
   При реализации статическим массивом память из-под элемента не освобождается, так как она выделена на стеке. В такой реализации просто сдвигается указатель выхода очереди на следующий элемент. При реализации списком при удалении память освобождается из-под узла элемента.
4. **Что происходит с элементами очереди при ее просмотре?**
   При просмотре очереди, как правило, элементы не изменяются.
   При просмотре очереди на основе массива мы просто обращаемся к элементам массива, используя индексы, которые представляют начало и конец очереди
   Для просмотра элементов очереди на основе списка мы начинаем с указателя на голову и перебираем все узлы списка, выводя их данные. Этот процесс также не меняет состояние очереди, так как мы лишь читаем данные, не изменяя их или не удаляя.
5. **От чего зависит эффективность физической реализации очереди?**
   Эффективность реализации зависит от вида выделения памяти, так как динамическое выделение памяти как правило замедляет программу, а также от размера дополнительных данных, таких как указатель на следующий элемент при реализации списком.
6. **Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?**
   Очередь массивом:
   Достоинства: Простота реализации, хорошая локальность данных.
   Недостатки: Фиксированный размер, что может привести к переполнению или избыточной памяти.

   Очередь списком:
   Достоинства: Динамический размер, удобство вставки и удаления в середине очереди.
   Недостатки: больше затраты памяти на хранение указателей, возможны фрагментация памяти.
7. **Что такое фрагментация памяти, и в какой части ОП она возникает?**
   В процессе моделирования очереди может оказаться, что при последовательных запросах на выделение и освобождении памяти под очередной элемент выделяется не та память, которая была только что освобождена при удалении элемента. Участки свободной и занятой памяти могут чередоваться, т.е. может возникнуть фрагментация памяти.
8. **Для чего нужен алгоритм «близнецов».**

Алгоритм "близнецов" используется в симуляции с двумя очередями. Он предполагает, что при появлении нового элемента, необходимо решить, в какую из двух очередей его поместить. Если одна из очередей короткая, новый элемент помещается в эту очередь, иначе выбирается случайная очередь. Такой подход позволяет достичь баланса между двумя очередями и равномерно распределить элементы между ними

9. **Какие дисциплины выделения памяти вы знаете?**

   First Fit (Первый Подходящий): выделяет первый блок памяти, который подходит по размеру.

   Best Fit (Лучший Подходящий): выделяет блок памяти, который наилучшим образом соответствует размеру запроса, минимизируя фрагментацию.

   Worst Fit (Худший Подходящий): выделяет самый большой из доступных блоков памяти, что может привести к уменьшению количества оставшейся свободной памяти.

   Next Fit (Следующий Подходящий): аналогичен First Fit, но начинает поиск с того места, где закончил предыдущий запрос.

   Quick Fit (Быстрый Подходящий): использует несколько списков фиксированных размеров для быстрого доступа к блокам нужного размера.

   Алгоритм "близнецов" (Twin Allocation)

10. **На что необходимо обратить внимание при тестировании программы?**

- Рассогласование между средними ожидаемыми временами и временами, полученными в моделирующей программе должно быть не больше 2 – 3%.
- Необходимо проверить правильность работы программы при различном заполнении очередей, т.е., когда время моделирования определяется временем обработки заявок и когда определяется временем прихода заявок;
- Отследить переполнение очереди, если очередь в программе ограничена.
- При реализации очереди списком необходимо тщательно следить за освобождением памяти при удалении элемента из очереди.
- Следует проверять, происходит ли при этом фрагментация памяти.

11. **Каким образом физически выделяется и освобождается память при динамических запросах?**

    Выделение памяти:

    При динамическом запросе памяти используется функция malloc, calloc. Когда программа вызывает одну из этих функций, операционная система выделяет участок памяти нужного размера. Эта память может быть неинициализированной (malloc) или проинициализированной нулями (calloc).

    Освобождение памяти:

Для освобождения динамически выделенной памяти используется функция free. Когда программа вызывает free для указателя, который ранее был возвращен malloc или calloc, операционная система помечает соответствующий блок памяти как свободный.