Отчет по 5 лабораторной работе

по предмету «Типы и Структуры Данных»

# группа ИУ7-34Б

# Елгин Илья Юрьевич

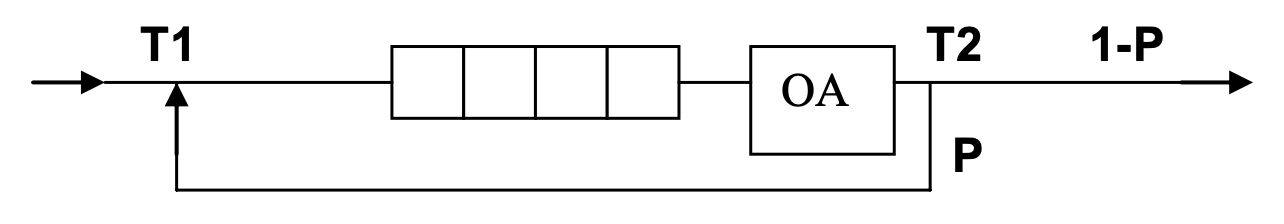
# Вариант 5

**Цель работы**

Отработка навыков работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка. Сравнительный анализ реализации алгоритмов включения и исключения элементов из очереди при использовании двух указанных структур данных. Оценка эффективности программы (при различной реализации) по времени и по используемому объему памяти.

**Задание (Вариант 5)**

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

**Входные данныe**

Цифры от 1 до 3 – выбор режима работы.

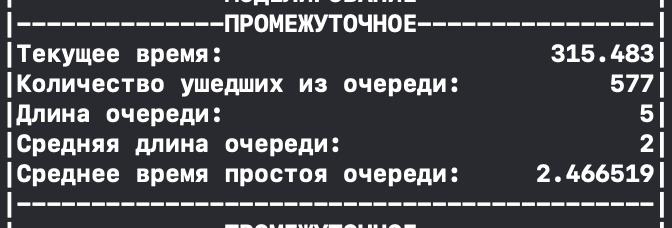
**Выходные данные**

После обслуживания каждых 100 заявок выводится информация о текущей и средней длине очереди, а в конце процесса - общее время моделирования и количество вошедших в систему и вышедших из нее заявок, данные о погрешностях и некоторые другие требуемые сведения о работе программы.

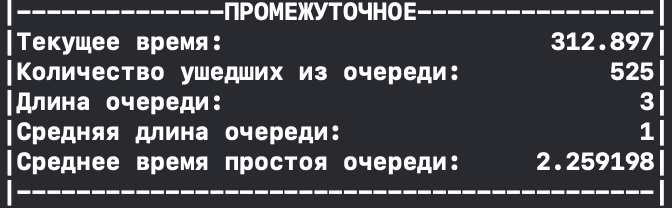
Адреса памяти удалённых и добавленных заявок при выборе соответствующего пункта меню.

**Интерфейс программы**

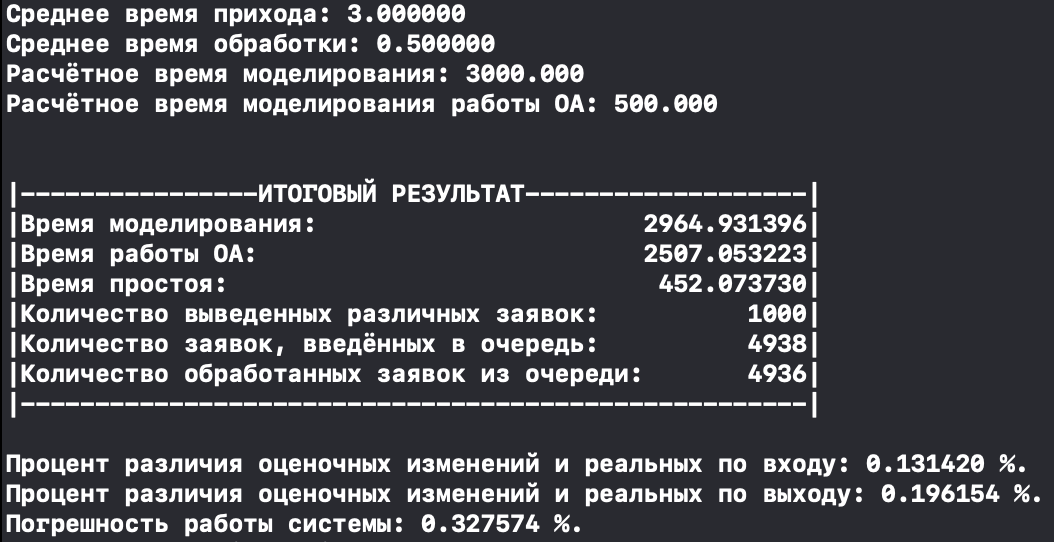
Данные об очереди при обработке первых (и каждых последующих) ста заявок первого типа. Реализация на массиве.



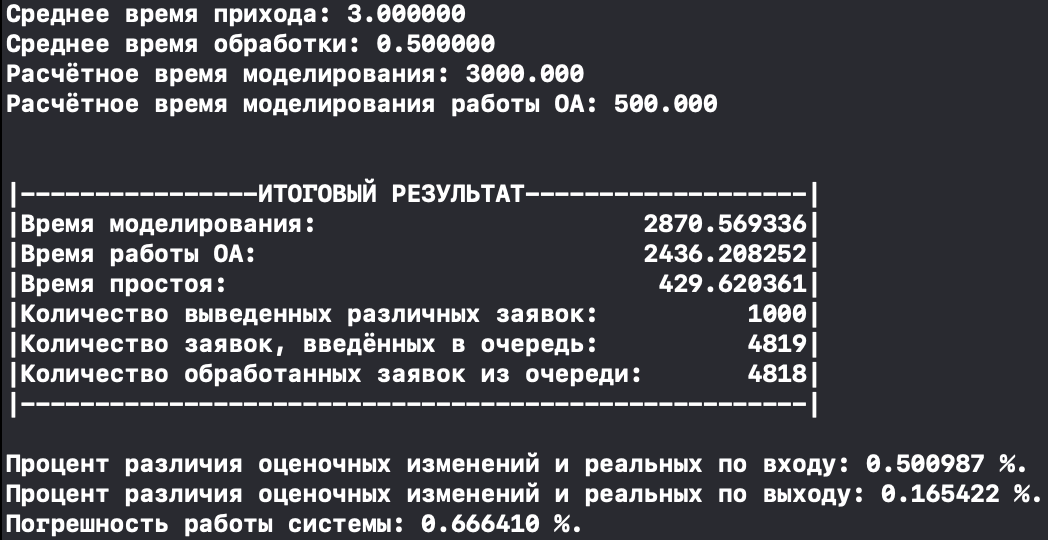
Реализация на списке.



Итоговый результат на массиве:



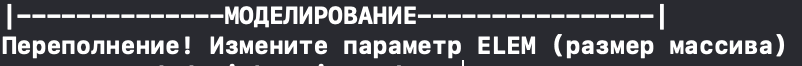
Итоговый результат на списке:



**Аварийные ситуации**

1. Переполнение при реализации массивом

Результат работы программы:



(Программа останавливает обработку заявок и не дает итогового результата при переполнении массива).

**Используемые структуры**

Структура для хранения очереди в виде массива:

typedef struct queue

{

float\* data\_store; //хранение заявок

int front; //индекс последнего элемента очереди

int max\_size; //максимальный размер массива

int num\_of\_elems; //количество элементов в очереди в данный момент

int summ\_of\_elems; //общее количество элементов

} queue\_t;

Структура узла списка:

**typedef struct qnode**

**{**

**struct qnode \*next; //указатель на следующий элемент списка**

**float time; //время обработки**

**} node\_t;**

Структура очереди в виде списка:

**typedef struct queue\_list**

**{**

**struct qnode \*pin; //указатель на конец списка**

**struct qnode \*pout; //указатель на начало списка**

**int num\_of\_elems; //количество элементов в очереди в данный момент**

**int summ\_of\_elems; //сумма элементов в целом**

**} queue\_t\_list;**

**Алгоритм:**

Запрашиваем у пользователя способ реализации очереди:

1: инициализируем очередь списком

2: инициализируем очередь массивом

Пока из очереди не вышли 1000 заявок:

Получить случайное время прихода заявки из интервала T1

Добавить это время к времени работы аппарата

Пока в очереди есть заявки и время работы аппарата положительное и из очереди не вышли 1000 заявок:

Получить случайное время обработки заявки из интервала T2

Вычесть это время из времени работы аппарата

Вытащить заявку из очереди

Если случайное число <= 1-P :

Проверить на переполнение и добавить заявку в конец очереди

Иначе увеличить количество вышедших заявок на 1

Если количество вышедших заявок кратно 100 то

Вывести статистическую информацию

Проверить на переполнение и добавить заявку в конец очереди

Вывести информацию о работе программы

Алгоритм вытаскивания элемента реализация списком:

Получить элемент как голову очереди

Передвинуть голову очереди на следующий элемент

Алгоритм добавления элемента реализация списком:

Выделить память под новый элемент списка

Записать в него данные

Установить указатель на следующий элемент хвоста очереди на этот элемент

Сделать новый элемент хвостом очереди

Алгоритм вытаскивания элемента реализация массивом:

Получить элемент как нулевой элемент массива

Сдвинуть все элементы (кроме нулевого) на 1 влево

Уменьшить количество элементов массива на 1

Алгоритм добавления элемента реализация массивом:

Записать в очередной свободный индекс массива данные

Увеличить количество элементов на 1

**Функции:**

*float randfrom(int min, int max);*

Выдаёт случайное число из заданного интервала

Входные данные: min- начало интервала, max- конец интервала.

Выходные данные: дробное случайное число

*float get\_time\_list(queue\_t\_list \*q1, automat\_list \*a);*

Производит моделирование работы аппарата и возвращает время его работы

Входные данные: q1- ссылка на очередь реализованную списком, a – ссылка на структуру для хранения статистических данных.

Выходные данные: дробное число время работы, изменение очереди и структуры для хранения статистических данных.

*float get\_time(queue\_t \*q1, automat \*a);*

Производит моделирование работы аппарата и возвращает время его работы

Входные данные: q1- ссылка на очередь реализованную массивом, a – ссылка на структуру для хранения статистических данных.

Выходные данные: дробное число время работы, изменение очереди и структуры для хранения статистических данных.

*int add\_elem(queue\_t \*queue, float item);*

Добавляет элемент в очередь.

Входные данные: queue – ссылка на очередь реализованную массивом, item – элемент добавляемый в очередь.

Выходные данные: Код ошибки, изменяет очередь.

*float delete\_elem(queue\_t \*queue);*

Вытаскивает очередной элемент из очередь.

Входные данные: queue – ссылка на очередь реализованную массивом,

Выходные данные: Значение элемента, изменяет очередь.

*int add\_elem\_list(queue\_t\_list \*queue, float time);*

Добавляет элемент в очередь.

Входные данные: queue – ссылка на очередь реализованную списком, item – элемент добавляемый в очередь.

Выходные данные: Код ошибки, изменяет очередь.

*float delete\_elem\_list(queue\_t\_list \*queueptr);*

Вытаскивает очередной элемент из очередь.

Входные данные: queue – ссылка на очередь реализованную массивом,

Выходные данные: Значение элемента, изменяет очередь.

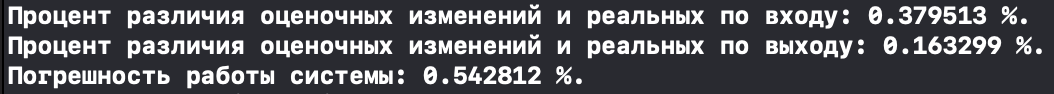
**Время моделирования**

Среднее время создания заявок: (T1\_max + T1\_min) / 2 = 3.

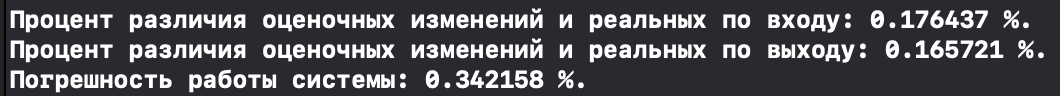
Среднее время обработки заявок (T2\_max + T2\_min) / 2 = 0.5.

1000 заявок выйдут из очереди за (max((T1\_max + T1\_min) / 2 \* 1000 , (T2\_max + T2\_min) / 2 \* 1000 / P)) = 3000 е.в.

Данные на одном из экспериментов для списка:



Для массива:



Погрешность в целом примерно равна в обеих реализациях, колеблется от 0.2% до 2%. При работе программы возникает фрагментация памяти. Несмотря на это, освобожденные адреса чаще всего все-таки опять занимаются, так как программе нужно очень большое количество адресов.

|  |  |  |  |
| --- | --- | --- | --- |
| Начальное время | Реализация | Время работы | Максимальный элемент |
| Т1 = 0 – 6  T2 = 0 - 1 | массив | 1мс | 6 |
| список | 4мс | 7 |
| T1 = 0 – 8  T2 = 5 - 6 | массив | 70мс | 5750 |
| список | 7мс | 5728 |
| T1 = 2 – 3  T2 = 1 - 4 | массив | 83мс | 7589 |
| список | 8мс | 7817 |
| T1 = 3 – 4  T2 = 0 - 2 | массив | Переполнение очереди | 11236 |
| список | 10мс | 15947 |

В данной программе вытаскивая из очереди элемент при реализации массивом происходит сдвиг всего массива, поэтому при больших очередях реализация очереди массивом уступает по времени реализации очереди списком. Если использовать циклический массив, в котором есть перемещающиеся указатели на голову и хвост очереди то массив начнёт выигрывать у списка по времени так как при представлении очереди в виде списка нужно освобождать и выделять память каждый раз, когда добавляется, или удаляется элемент, а это времезатратно.

Также к недостаткам очереди-списка можно отнести возникновение фрагментации памяти и большее количество памяти для хранения указателей.

**Вывод:**

Очередь позволяет получать, хранить и обрабатывать элементы в порядке их поступления. Её удобно использовать при обработке заявок и прочих элементов, скорость поступления которых выше чем скорость их обработки. При известном максимальном количестве элементов эффективнее очередь реализовывать с помощью массива, так как он даёт преимущество по времени, список же лучше использовать, когда максимальное количество элементов неизвестно.

При реализации очереди массивом эффективнее сделать его зацикленным, тоесть хранить индексы головы и хвоста.

**Ответы на вопросы**

1. Что такое очередь?

Очередь – это последовательный список переменной длины, включение элементов в который идет с одной стороны, а исключение – с другой стороны.

1. Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?

В списке память под элемент очереди выделяется непосредственно в процессе его добавления. Объем памяти, который занимает очередь, изменяется в процессе выполнения программы и напрямую зависит от количества элементов в очереди в каждый момент времени. При реализации очереди массивом выделяется последовательная область памяти константного размера. Выделение памяти происходит в начале работы программы. При необходимости память перевыделяется.

1. Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?

В массиве освобождение памяти происходит в конце работы программы (или при удалении очереди). При удалении элемента из очереди происходит только смещение указателя. В списке при удалении элемента из очереди происходит освобождение памяти, которая была выделена под этот элемент.

1. Что происходит с элементами очереди при ее просмотре?

При просмотре очереди хвостовой элемент из нее удаляется.

1. Каким образом эффективнее реализовывать очередь. От чего это зависит?

Если необходимо избежать фрагментации памяти, то лучше использовать очередь на массиве. Однако такой способ лучше использовать в том случае, если заранее известно количество элементов в очереди. Иначе лучше реализовывать список.

1. В каком случае лучше реализовать очередь посредством указателей, а в каком – массивом?

Очередь лучше реализовывать с помощью указателей, если новые элементы в среднем появляются реже, чем происходит полное очищение очереди – в общем случае фрагментация не возникает.

1. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?

При реализации очереди массивом не возникает фрагментации памяти, однако может произойти переполнение очереди, а также затрачивается дополнительное время на сдвиг элементов (можно исключить сдвиг используя кольцевой массив). При реализации очереди списком затрачивается большее количество времени при добавлении нового элемента, для хранения указателей требуется дополнительная память.

1. Что такое фрагментация памяти?

При последовательных запросах на выделение и освобождении памяти под элемент не всегда выделяется память, которая была только что освобождена.

1. На что необходимо обратить внимание при тестировании программы?

При тестировании программы необходимо обратить внимание на переполнение очереди, фрагментацию памяти при реализации очереди списком.

1. 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. Попытка считать данные из этого блока может привести к ошибке программы или неверному результату, поскольку они могут быть уже изменены.