**Отчет**

**Лабораторная работа №5**

“Обработка очередей”

Выполнила:

студент группы ИУ7-35Б

Лаврова Анастасия

**Цель работы:**

Приобрести навыки работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка, провести сравнительный анализ реализации алгоритмов включения и исключения элементов из очереди при использовании указанных структур данных, оценить эффективности программы по времени и по используемому объему памяти.

**Условие задачи:**

Заявки 1-го и 2-го типов поступают в "хвосты" своих очередей по случайному закону с интервалами времени **Т1** и **Т2**. В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за времена **Т3** и **Т4**, после чего покидают систему.

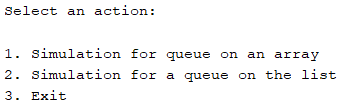
Заявка 2-го типа может войти в ОА, если в системе нет заявок 1-го типа. Если в момент обслуживания заявки 2-го типа в пустую очередь входит заявка 1-го типа, то она немедленно поступает на обслуживание; обработка заявки 2-го типа прерывается и она возвращается в "хвост" своей очереди.

Смоделировать процесс обслуживания первых 1000 заявок **1-го типа**, выдавая после обслуживания каждых 100 заявок **1-го типа** информацию о текущей и средней длине каждой очереди, а в конце процесса – общее время моделирования и количестве вошедших в систему и вышедших из нее заявок обоих типов, среднем времени пребывания заявок в очереди, количестве «выброшенных» заявок второго типа. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

**Техническое задание**

**1) Входные данные:**

Программа позволяет выбрать структуру данных для хранения очереди. 1 – моделирование для очереди на массиве. 2 – моделирование для очереди на списке.



**2) Выходные данные:**

Текущая и средняя длины каждой очереди. Количество вошедших в систему и вышедших из нее заявок обоих типов, количество «выброшенных» заявок второго типа. Время «простоя», процентная оценка погрешности работы программы (проверка правильности работы по входу и выходу), общее время моделирования, время выполнения программы. Адреса элементов очереди при реализации очереди списком.

**Описание задачи, реализуемой программой:**

1. Реализация очереди в виде массива (кольцевая очередь).

2. Реализация очереди в виде списка.

3. Смоделировать процесс обслуживания заявок с выводом состояния очередей.

5. Определение текущей и средней длины очереди.

6. Теоретический расчет времени моделирования, время простоя, количества вошедших и вышедших заявок, время работы ОА.

7. Оценка погрешности полученных результатов при реализации.

**Алгоритм**

1. Реализация очереди 1 – с помощью массива 2 – с помощью списка

2. Если 2: запрос, нужен ли пользователю вывод информации о памяти.

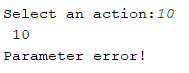
3. Моделирование процесса обслуживания очереди с выводом промежуточных результатов.

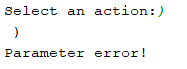
**Способ обращения к программе:**

Через консоль

**Аварийные ситуации:**

Некорректный параметр меню





Переполнение очереди



**Используемые структуры данных:**

Дескриптор очереди

**typedef struct** queue  
{  
 **char** name[30]; *// Имя очереди* **void**\* low; *// Адрес нижней границы* **void**\* up; *// Адрес верхней границы* **void**\* pin; *// Указатель на "хвост"* **void**\* pout; *// Указатель на "голову"* **int** max\_num; *// Число элементов* size\_t size; *// Размер типа данных* **int** count\_req; *// Число запросов* **int** sum\_size; *// Средняя длина* **int** tmp\_size; *// Текущая длина* **int** sum\_time; *// Общее время* **int** out\_req; *// Число запросов на выход* **int** in\_req; *// Число запросов на вход*} queue;

Структура для списка

**typedef struct** node node;  
**struct** node  
{  
 **char** inf;  
 node \*next;  
};

Массив

que1 = (**char** \*)malloc(d1.max\_num \* **sizeof**(**char**));

Список

node\* que1 = **NULL**;

**Функции:**

**void** showLen(queue\* q);

Вход: очередь q.

Выход: информация об общей и средней длине очереди.

**void** workArray(**int** n, **int** interval, interval\_time t1, interval\_time t2, interval\_time t3, interval\_time t4, **int** flag);

Вход: временные интервалы, флаг.

Выход: моделирование очереди на массиве.

**void** workList(**int** n, **int** interval, interval\_time t1, interval\_time t2, interval\_time t3, interval\_time t4, **int** flag);

Вход: временные интервалы, флаг.

Выход: моделирование очереди на списке.

**double** getTime(interval\_time t);

Вход: временной интервал.

Выход: время.

node\* createElem(**char** c);

Вход: элемент char.

Выход: создание элемента списка.

node\* pushFront(node \*head, node \*elem);

Вход: голова списка и элемент для добавления.

Выход: добавление элемента.

node\* popFront(node \*\*head);

Вход: голова списка.

Выход: удаление с головы.

node\* popEnd(node \*\*head);

Вход: голова списка.

Выход: удаление с хвоста.

**void** freeAll(node \*head);

Вход: голова списка.

Выход: освобождение памяти.

node\* ListPush(node\* q, **char** c, node\*\* used\_memory, **int** \*count\_used, node\*\* freed\_memory, **int**\* count\_freed, **int**\* second\_used);

Вход: очередь, элемент для добавления, использованная память, количество использованной памяти, освобожденная память, количество освобожденной памяти, количество повторно использованных адресов.

Выход: добавление элемента в очередь.

node\* ListPop(node\*\* q);

Вход: очередь.

Выход: удаление элемента из очереди.

**void** ListPrint(node\* q);

Вход: очередь.

Выход: вывод очереди на экран.

**void** ArrayPush(queue\* q, **char**\* elem, **char** c);

Вход: очередь, элемент для добавления.

Выход: добавление элемента.

**char** ArrayPop(queue\* q, **char**\* elem);

Вход: очередь, элемент для удаления.

Выход: удаление элемента.

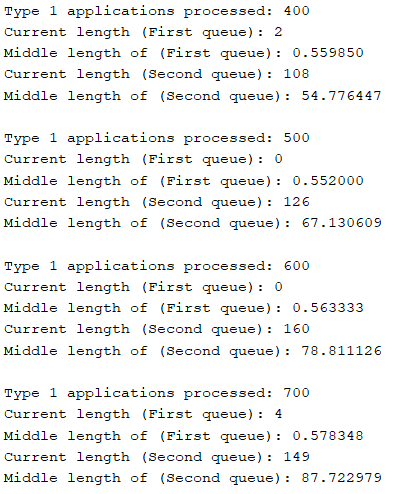
**void** ArrayPrint(queue\* q, **char**\* elem);

Вход: очередь.

Выход: вывод очереди на экран.

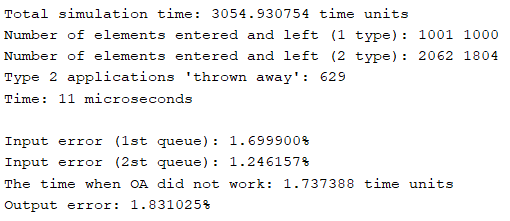
**Тесты:**

Промежуточный вывод

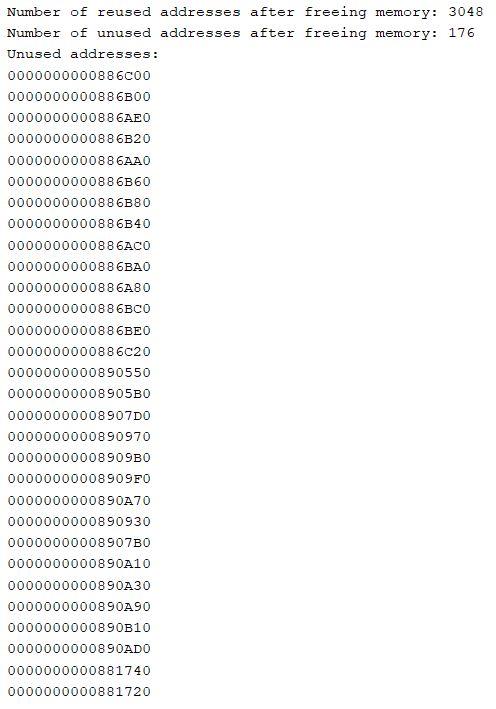


Результат моделирования:

res = n \* max(среднее время входа первой заявки, среднее время выхода первой заявки) = 3000



Вывод информации о памяти



**Оценка эффективности:**

На массиве

Использованная память: 32080 байт

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Число заявок  1-го типа | Число заявок 2-го типа | Время моделирования (е.в.) | Время работы (мкс) |
| 1 | 1000 | 1923 | 2958.845103 | 458 |
| 2 | 1000 | 1850 | 3022.304049 | 464 |
| 3 | 1000 | 2102 | 3025.672660 | 463 |
| 4 | 1000 | 1910 | 3025.721911 | 519 |
| 5 | 1000 | 2056 | 3077.698157 | 462 |
| 6 | 1000 | 2083 | 3049.225118 | 544 |
| 7 | 1000 | 2045 | 2954.321242 | 440 |
| 8 | 1000 | 2037 | 2934.372229 | 465 |
| 9 | 1000 | 2236 | 2990.872352 | 474 |
| 10 | 1000 | 2204 | 3004.323260 | 447 |
| Среднее | 1000 | 12044 | 3004.335608 | 473.6 |

На списке

Использованная память: 61224 байт

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Число заявок  1-го типа | Число заявок 2-го типа | Время моделирования (е.в.) | Время работы (мкс) |
| 1 | 1000 | 1834 | 3024.785534 | 3240 |
| 2 | 1000 | 1952 | 3016.822036 | 2820 |
| 3 | 1000 | 1902 | 3069.367394 | 3159 |
| 4 | 1000 | 2129 | 2998.525696 | 2593 |
| 5 | 1000 | 1916 | 2995.933582 | 3035 |
| 6 | 1000 | 1963 | 3038.888007 | 2069 |
| 7 | 1000 | 1904 | 3029.712658 | 2783 |
| 8 | 1000 | 1995 | 2964.229659 | 3562 |
| 9 | 1000 | 2150 | 3052.744417 | 2470 |
| 10 | 1000 | 2047 | 2929.497138 | 3313 |
| Среднее | 1000 | 1979.2 | 3012.050612 | 2904.4 |

**Вывод:**

При представлении очереди в виде списка используется большее количество памяти для хранения указателей. К недостаткам очереди-списка можно отнести возникновение фрагментации памяти. Очередь-список позволяет воспользоваться памятью, ограниченной лишь объемом оперативной памяти компьютера.

Эффективность реализации работы со списком зависит от кол-ва операций добавления и извлечения элемента из очереди. Недостатком очереди-списка является то, что при совершении этих операций он обязательно выделяет или освобождает память.   
Очередь массив производит операции выделения/освобождения намного реже.

**Контрольные вопросы**

*1. Что такое очередь?*

Очередь – последовательной список переменной длины. Включение элементов идёт с «хвоста» списка, исключение – с «головы» списка. Принцип работы: первым пришёл – первым вышел, «First In First Out».

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

При реализации очереди списком, под каждый новый элемент выделяется sizeof(element) + 4 байт памяти из кучи, элементы связываются указателями. При реализации очереди массивом, выделяется блок памяти из N\*sizeof(element) байт, где N – максимальное количество элементов в очереди, элементы следуют друг за другом последовательно.

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

При реализации очереди списком, головной элемент считывается, указатель на «голову» очереди переходит на следующий элемент, считанный элемент удаляется.

Память, выделенная под массив, очищается в конце работы программы.

*4. Что происходит с элементами очереди при ее просмотре?*

При просмотре очереди (в классической реализации), головной элемент из неё удаляется, указатель p\_out сдвигается на следующий элемент.

*5. Каким образом эффективнее реализовывать очередь. От чего это зависит?*

При реализации очереди списком, проще всего добавлять и удалять из неё элементы, однако может возникнуть фрагментация памяти. При реализации очереди массивом фрагментации не возникает, однако может возникнуть переполнение памяти. Также для массива необходимо знать тип данных. Но операции добавления и исключения элементов быстрее работают на массиве. Способ реализации зависит от того, в чем мы больше ограничены – в памяти или во времени выполнения операций

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

Массив эффективнее использовать, когда важна скорость, так как действия с массивом выполняются быстрее. Список выигрывает в эффективности по памяти, так как количество элементов списка ограничивается только объемом оперативной памяти.

*7. 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 (в обычной реализации). Сдвига можно избежать, если использовать кольцевой массив, однако при этом усложняются операции добавления и удаления элементов. При реализации списком проще всего реализуются алгоритмы добавления и удаления элементов, но может возникнуть фрагментация.

*8. Что такое фрагментация памяти?*

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.

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

При реализации очереди списком необходимо следить за освобождением памяти при удалении элемента из очереди. Если новые элементы приходят чаще, чем удаляются старые, очередь растёт и может происходить фрагментации памяти.

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

Программа даёт 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 неопределенному поведению, поскольку они могут быть уже изменены.