**Отчёт лабораторной работы №5**

по курсу «Типы и структуры данных»

Обработка очередей

Отчёт выполнила:

Кондрашова Ольга

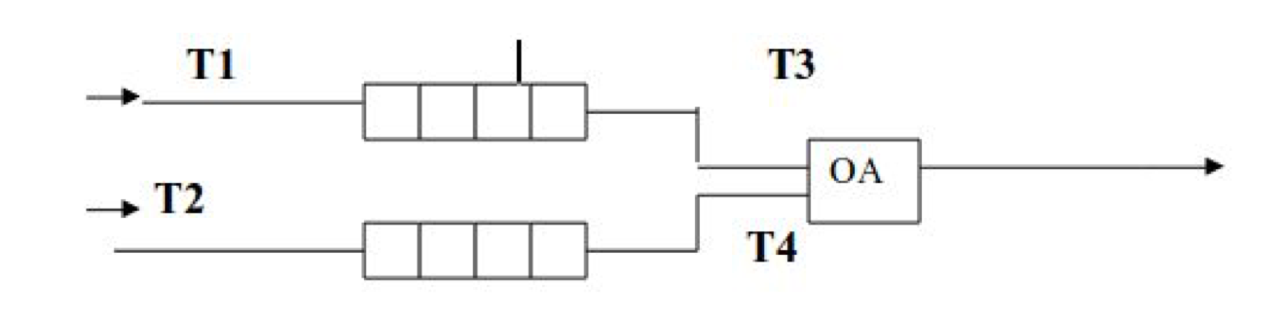
Группа ИУ7-35Б

Вариант 9

**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 6**:

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и двух очередей заявок двух типов.  
  
Заявки 1-го и 2-го типов поступают в "хвосты" своих очередей по случайному закону с интервалами времени Т1 и Т2, равномерно распределенными от 1 до 5 и от 0 до 3 единиц времени (е.в.) соответственно. В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за времена Т3 и Т4, распределенные от 0 до 4 е.в. и от 0 до 1 е.в. соответственно, после чего покидают систему. (Все времена – вещественного типа) В начале процесса в системе заявок нет.

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 (система с абсолютным приоритетом и повторным обслуживанием).

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.

**Входные данные:**

Программа позволяет выбрать структуру данных для хранения очереди.

1 – моделирование для очереди на массиве.  
2 – моделирование для очереди на списке.

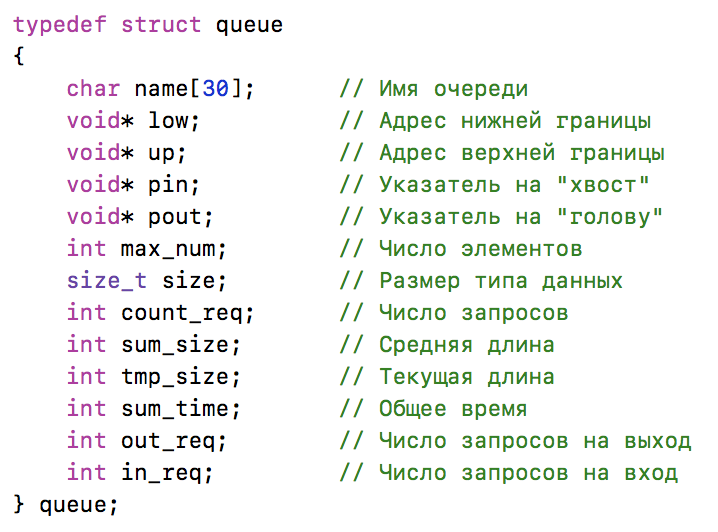
**Выходные данные:**

После обслуживания каждых 100 заявок 1-ого типа выводится информация о текущей и средней длине каждой очереди, а в конце процесса - общее время моделирования, количество вошедших в систему и вышедших из нее заявок обоих типов, количество «выброшенных» элементов второго типа, процентная оценка погрешности программы.

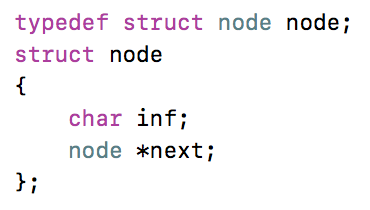
**Обращение к программе:**Через консоль

**Внутренняя структура данных:**

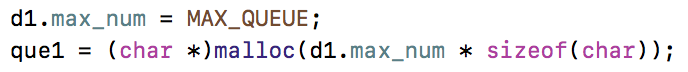
Дескриптор очереди:



Структура для списка:



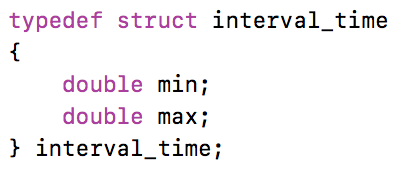
Очередь-массив:

MAX\_QUEUE – максимальный размер очереди.  
  


Очередь-список:



Структура для временных интервалов:



**Описание задачи, реализуемой программой**

1. Реализация очереди в виде массива.
2. Реализация очереди в виде списка.
3. Моделирование процесса обслуживания очереди с выводом промежуточных результатов.
4. Оценка погрешности.

**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.

**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\* queueListPush(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\* queueListPop(node\*\* q)

Вход: очередь.

Выход: удаление элемента из очереди.

void queueListPrint(node\* q)

Вход: очередь.

Выход: вывод очереди на экран.

void queueArrayPush(queue\* q, char\* elem, char c)

Вход: очередь, элемент для добавления.

Выход: добавление элемента.

char queueArrayPop(queue\* q, char\* elem)

Вход: очередь, элемент для удаления.

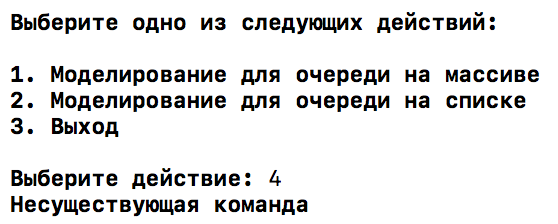
Выход: удаление элемента.

void queueArrayPrint(queue\* q, char\* elem)

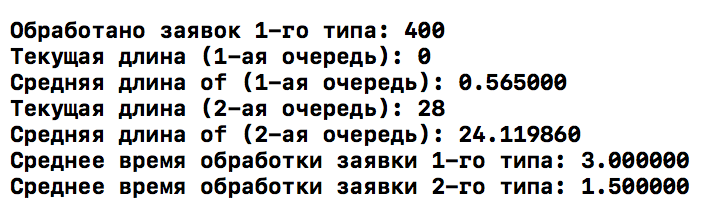
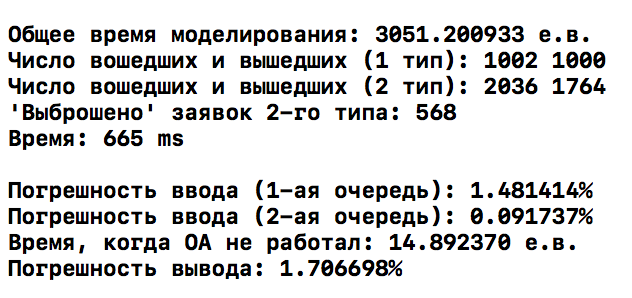
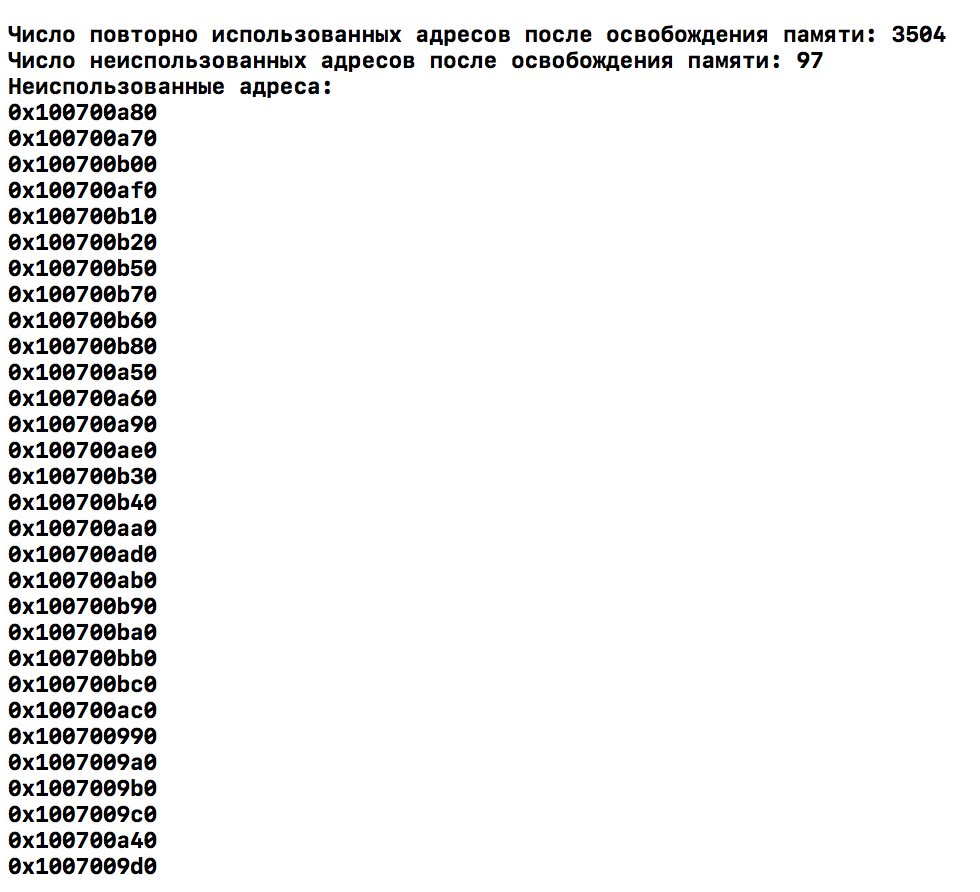
Вход: очередь.

Выход: вывод очереди на экран.

**Аварийные ситуации и ошибки пользователя**

1. Выбор несуществующей опции  
   

**Тесты**

1. Промежуточный вывод  
   
2. Результат моделирования  
   
3. 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 | 2017.6 | 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:**

Время моделирования очереди на массиве в среднем быстрее на 0.266%.

Время работы очереди на массиве в среднем быстрее на 16%.

При представлении очереди в виде списка используется большее кол-во 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 операцию выделения памяти всего один раз.

**Ответы на 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 сколько памяти выделяется под хранение очереди при различной его 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 различной реализации стека?

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

При реализации очереди массивом, головной элемент считывается; чтобы не случалось переполнения, при каждом исключении из очереди обычно производят сдвиг всех элементов к «голове». Но на эти последовательные сдвиги затрачивается время. Поэтому более эффективно использовать кольцевуюочередь. 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 pout сдвигается на следущий 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?

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

Очередь лучше реализовывать с помощью массива, если важна эффективность по времени. Очередь-список лучше использовать, если важна память, т.к. огчередь-список позволяет воспользоваться памятью, ограниченной лишь объемом оперативной памяти компьютера.

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. 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 непредвиденным последствиям, поскольку они могут быть уже изменены.