Отчет по 5 лабораторной работе по предмету «Типы и Структуры Данных»

# Подготовила студентка группы ИУ7-25Б

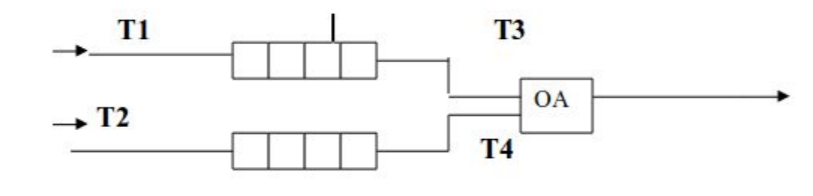
# Маргарита Мищенко

# Вариант 6

**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 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. Отчет по лабораторной работе должен содержать выводы о том, в каких случаях применение какой структуры данных более целесообразно для реализации очереди, какую выгоду дает та или иная реализация. Выводы должны быть подтверждены результатами числовых сравнений расходования памяти и времени выполнения схожих операций.

**Исходные данныe**

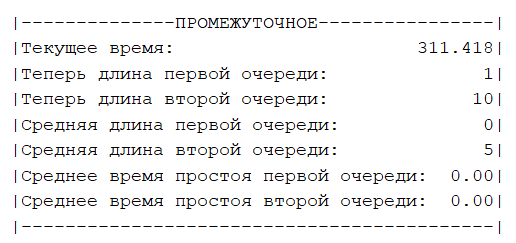
Исходные данные в данной программе не требуются, для изменения параметров следует изменить данные в файле constants.h, чтобы выбрать вид реализации (массив или список), нужно запустить одну из двух реализаций проекта (они реализованы как отдельные программы). Чтобы включить показывание адресов при добавлении и удалении элемента из и в очередь, необходимо расскомментировать строку “#define LOG” в проекте со списком.

**Выходные данные**

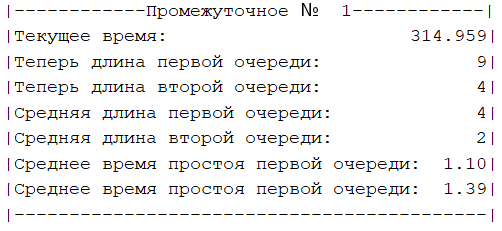
После обслуживания каждых 100 заявок 1-ого типа выводится информация о текущей и средней длине каждой очереди, а в конце процесса - общее время моделирования и количество вошедших в систему и вышедших из нее заявок обоих типов, данные о погрешностях и некоторые другие требуемые сведения о работе программы.

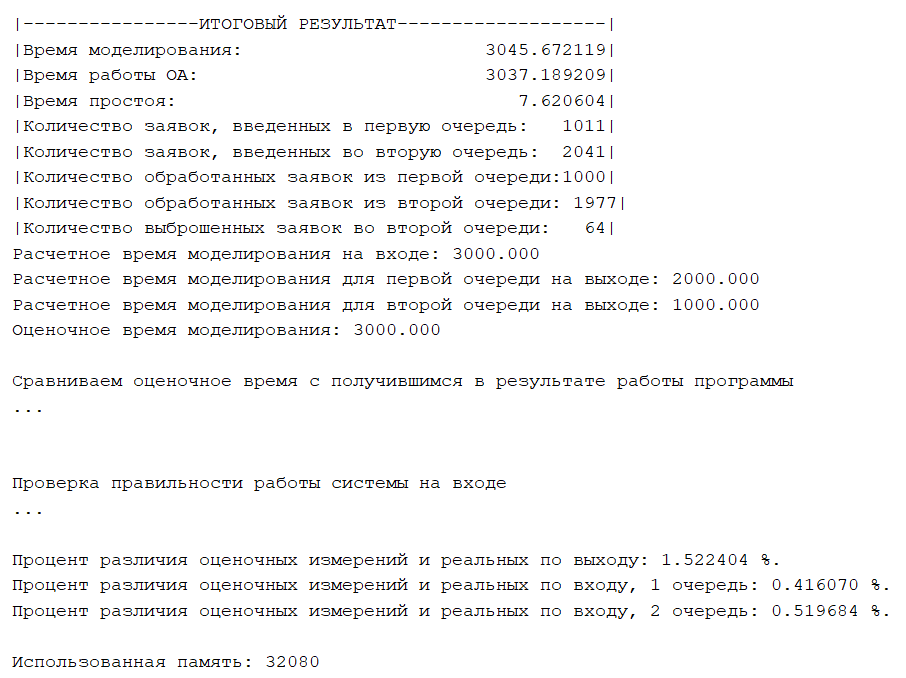
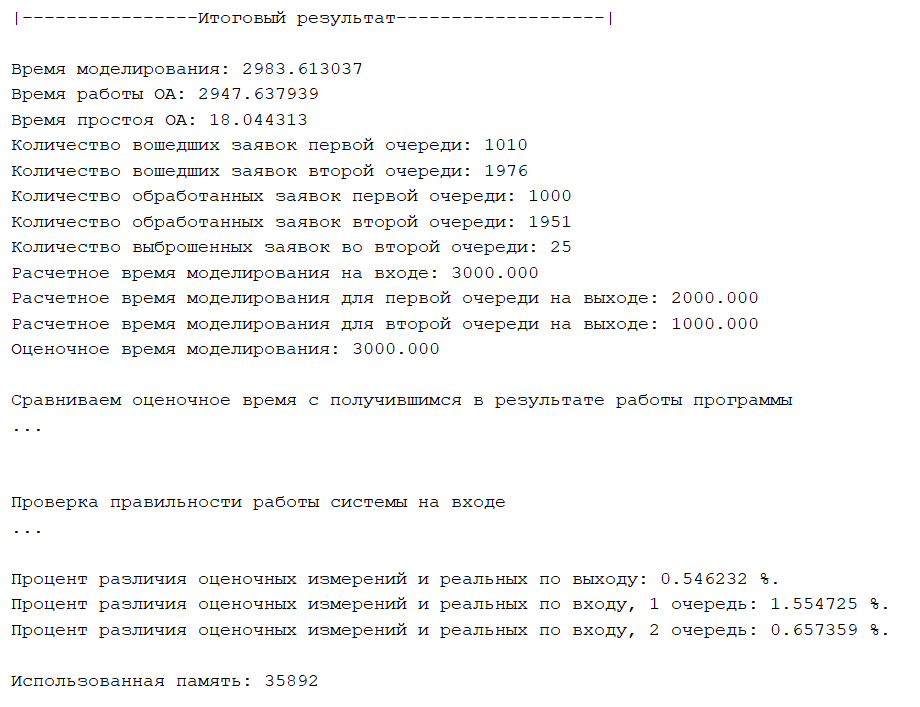
**Интерфейс программы**

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



Реализация на списке.

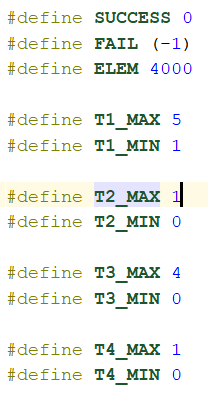


Итоговый результат на массиве: Итоговый результат на списке: 

**Аварийные ситуации**

1. Переполнение при реализации массивом

Пример переполнения: параметры



Результат работы программы:



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

**Используемые структуры**

При реализации массивом:

Структура содержащая тип заявок

**typedef enum** req\_type  
{  
 ***NONE***, ***T1***, ***T2***}type\_t;  
  
Структура для хранения очереди

**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 в очереди  
} queue\_t;

При реализации списком:

Структура содержащая тип заявок

**typedef enum** req\_type  
{  
 ***NONE***,  
 ***T1***,  
 ***T2***}type\_t;

Структура узла списка  
  
**typedef struct** qnode  
{  
 **struct** qnode \*next; - указатель на следующий элемент списка  
 **enum** req\_type type; - тип заявки  
 **float** time; - время обработки  
} node\_t;

Структура очереди  
  
**typedef struct** queue  
{  
 **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 очереди в целом  
} queue\_t;

**Время моделирования**

Среднее время создания заявок:

1-го типа: (5 + 1) / 2 = 3 е.в;

2-го типа: (3 + 0) / 2 = 1.5 е.в.

Среднее время обработки заявок

1-го типа: (4 + 0) / 2 = 2 е.в.

2-го типа: (1 + 0) / 2 = 0.5 е.в.

1000 заявок первого типа создается за 3 \* 1000 = 3000 е.в.. За это время будет создано около 3000е.в / 1.5 = 2000 заявок 2 типа.

1000 заявок первого типа будет обработано за 1000 \* 2 = 2000 е.в..

2000 заявок второго типа будет обработано за 2000 \* 0,5 = 1000 е.в

При данных параметрах время создания заявок обоих типов больше, чем время их обработки, следовательно, как и указано в методических указаниях, на время моделирования влияет только время создания заявок 1-го типа. Таким образом, получаем, что среднее время моделирования 3000 е.в..

**Сравнение эффективности**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № (список) | Число заявок первого типа | Число заявок второго типа | Время моделирования (в единицах времени) | Время работы, с |
| 1 | 1004 | 2065 | 3036 | 0.004 |
| 2 | 1019 | 1980 | 3018 | 0.005 |
| 3 | 1034 | 2067 | 3086 | 0.007 |
| 4 | 1007 | 2007 | 3005 | 0.002 |
| 5 | 1003 | 1975 | 3070 | 0.002 |
| 6 | 1009 | 2101 | 3138 | 0.004 |
| 7 | 1022 | 2014 | 2970 | 0.005 |
| Среднее | 1014 | 2029 | 3046 | 0.004 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № (массив) | Число заявок первого типа | Число заявок второго типа | Время моделирования (в единицах времени) | Время работы, с |
| 1 | 1013 | 2043 | 3062 | 0.002 |
| 2 | 1002 | 1942 | 2919 | 0.002 |
| 3 | 1011 | 1994 | 2981 | 0.001 |
| 4 | 1007 | 2026 | 2991 | 0.002 |
| 5 | 1007 | 1970 | 2990 | 0.001 |
| 6 | 1002 | 2010 | 2981 | 0.001 |
| 7 | 1002 | 2023 | 3026 | 0.000 |
| Среднее | 1006 | 2001 | 2992 | 0.00128 |

Данные на одном из экспериментов для списка:

Процент различия оценочных измерений и реальных по выходу: 0.983447 %.

Процент различия оценочных измерений и реальных по входу, 1 очередь: 3.215066 %.

Процент различия оценочных измерений и реальных по входу, 2 очередь: 1.700168 %

Использованная память: 36492

Для массива:

Процент различия оценочных измерений и реальных по выходу: 0.873096 %.

Процент различия оценочных измерений и реальных по входу, 1 очередь: 0.667270 %.

Процент различия оценочных измерений и реальных по входу, 2 очередь: 0.274508 %.

Использованная память: 32080

Погрешность в целом примерно равна в обеих реализациях, колеблется от 0.1% до 3.5%. Время выполнения программы с использованием очереди на основе списка в среднем примерно в 2.5 раз больше реализации с помощью массива. При работе программы возникает фрагментация. Несмотря на это, освобожденные адреса чаще всего все-таки опять занимаются, так как программе нужно очень большое количество адресов.

При представлении очереди в виде списка используется большее количество памяти для хранения указателей (в представленном примере массив на 12% более эффективен по памяти). Также к недостаткам очереди-списка можно отнести возникновение фрагментации памяти. По времени список тоже менее эффективен, так как он должен освобождать и выделять память каждый раз, когда добавляется, или удаляется элемент, а это времезатратно.

**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. 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.