|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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**

**«ОБРАБОТКА ОЧЕРЕДЕЙ»**

Студент Ву Минь Куанг

Группа ИУ7И – 34Б

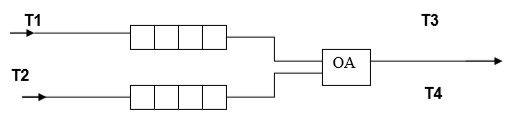
Преподаватель Силантьева А. В.

*2021 г.*

**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.

**Условие задачи (4 вариант):**

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 любого типа может войти в ОА, если:

* а) она вошла в пустую систему;
* б) перед ней обслуживалась заявка ее же типа;
* в) перед ней из ОА вышла заявка другого типа, оставив за собой пустую очередь (система с чередующимся приоритетом).

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа, выдавая после обслуживания каждых 100 заявок информацию 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. По требованию пользователя выдать на экран адреса элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

**Входные данные:**Интервалы времени, число заявок первого типа для обработки, через какой промежуток заявок сохранять лог.

**Выходные данные:**Лог, краткая сводка по процессу.

**Функция программы:**Реализация системы обработки заявок из двух очередей, с чередующимся приоритетом.

Обращение к программе: через консоль командой ./main.exe.

**Структуры данных:**

Очередь массивом:

**char** \*q1, \*q2;

Очередь списком:

node \*q1 = **NULL**, \*q2 = **NULL**;

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

Дескриптор:

**struct** discriptor{  
 **void**\* low;  
 **void**\* up;  
 **void**\* p\_in;  
 **void**\* p\_out;  
 **int** max\_num;  
 **int** count\_request;  
 **int** sum\_size;  
 **int** curr\_size;  
 **int** out\_request;  
 **int** in\_request;  
};

Интервалы времени:

**struct** time\_range{  
 **double** min;  
 **double** max;  
};

**Функции:**

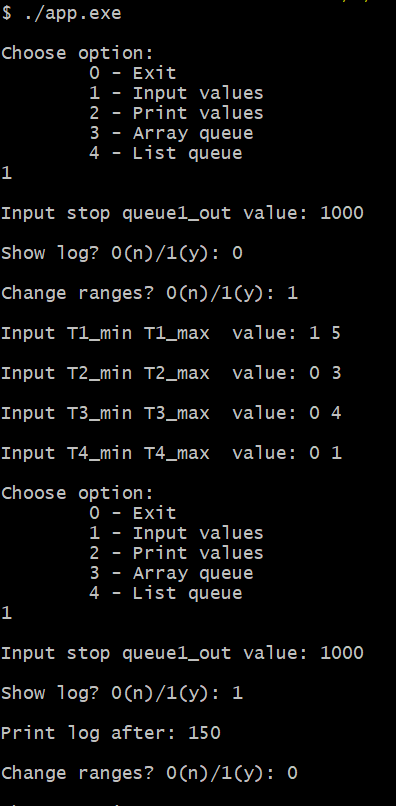
Функции для очереди массивом:

**void** arr\_push(discriptor\* d, **char**\* qu, **char** c);  
**char** arr\_pop(discriptor\* d, **char**\* qu);  
**void** arr\_print(discriptor\* d, **char**\* qu);  
  
**void** go\_array(**int** n, **int** interval, time\_range t1, time\_range t2, time\_range t3, time\_range t4, **int** log\_flag);

Функции для очереди списком:

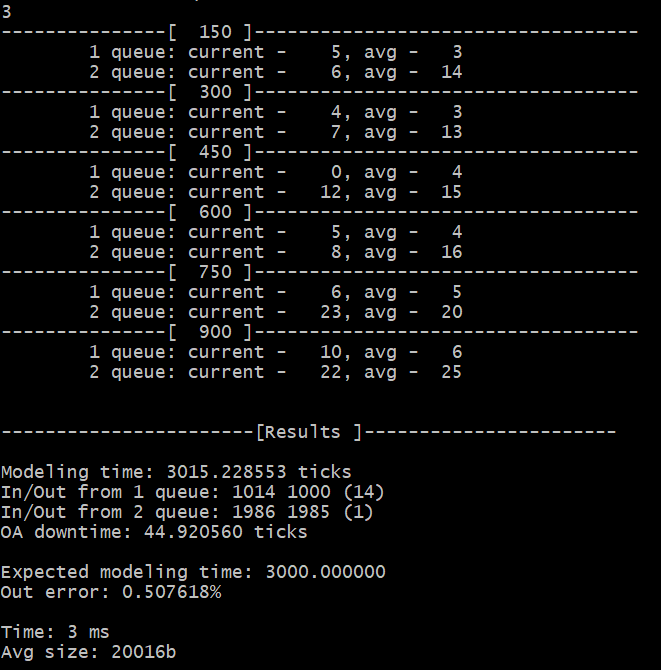
node\* create\_node(**char** c);  
node\* add\_node(node \*head, node \*item);  
node\* pop\_node(node \*\*head);  
  
**void** free\_all(node \*head);  
  
node\* list\_push(node\* qu, **char** c, node\*\* used\_memory, **int** \*count\_used, node\*\* freed\_memory, **int**\* count\_freed, **int**\* second\_used);  
node\* list\_pop(node\*\* qu);  
**void** list\_print(node\* qu);  
  
**void** go\_list(**int** n, **int** interval, time\_range t1, time\_range t2, time\_range t3, time\_range t4, **int** log\_flag);

**Интерфейс:**

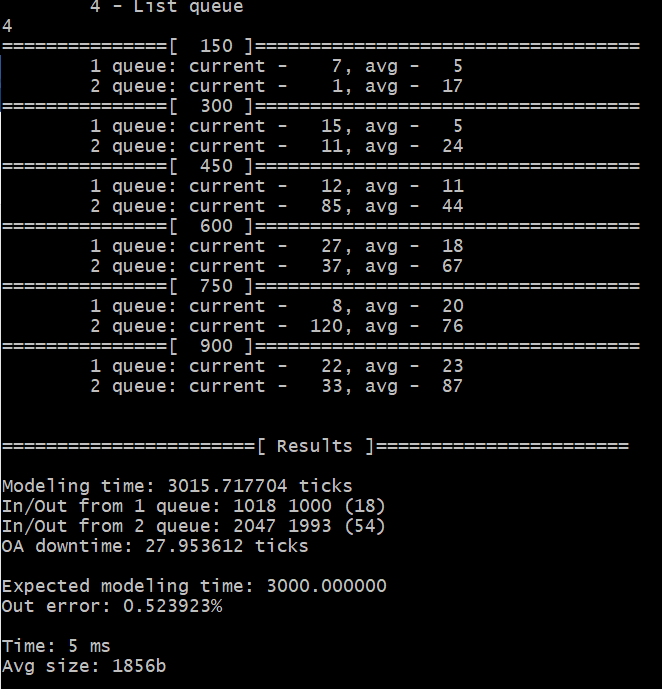
Изменение значений: ****

Работа программы:

(Очередь массивом)



(Очередь списком)

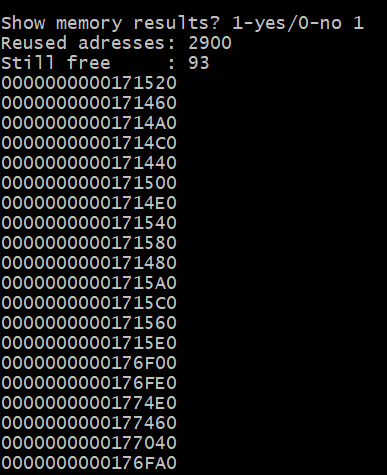


* (1 + 5) / 2 \* 1000 = 3000 на добавление 1000 первого типа

За это время добавится 3000 / ((0 + 3) / 2) = 2000 заявок второго типа

Обработаются эти заявки за 1000 \* ((0 + 4) / 2) + 2000 \* ((0 + 1) / 2) = 3000 ев  
3000ев = 3000ев значит OA работает без простоя и общее время моделирования 3000ев

**Фрагментация памяти:**



При некорректном вводе программа выдаст ошибку.

**Анализ эффективности (по памяти и времени):**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 000 | 10 000 | 100 000 |
| Массив | <1 (20008b) | 5 (20008b) | 50 (20008b) |
| Список | 2 (648) | 24 (1288b) | 1510 (7688b) |

\*время в ms

Размер списка сильно варьируется.

**Вывод:** эффективнее использовать массив, он дает заметный выигрыш по времени при больших размерах очереди. При представлении очереди в виде списка используется большее количество памяти для хранения указателей. По времени список тоже менее эффективен, так как он должен освобождать и выделять память каждый раз, когда добавляется, или удаляется элемент, а это времязатратно. Однако, если нам заранее не известен хотя бы примерный объем данных, то лучше использовать список, т.к. выделять непрерывную область памяти, которую требует массив, не всегда возможно.

**Контрольные вопросы:**

1. **Что такое очередь?**

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

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

При хранении кольцевым массивом: кол-во элементов \* размер одного элемента очереди. Память выделяется на стеке при компиляции, если массив статический. Либо память выделяется в куче, если массив динамический. При хранении списком: кол-во элементов \* (размер одного элемента очереди + указатель на следующий элемент). Память выделяется в куче для каждого элемента отдельно.

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

При реализации списком, считывается первый с головы (текущий) элемент, происходит смещение головы, а тот элемент освобождается.

При реализации очереди массивом, считывается текущий элемент, остальные элементы сдвигаются на 1 элемент в сторону текущего элемента.

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

При просмотре очереди текущий элемент из нее удаляется.

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

Выбор способа зависит от приоритетов: время или память.

При реализации списком легче добавить и удалить элемент, но при этом может возникнуть фрагментация памяти. При реализации массивом при удалении необходимо сдвигать все его элементы, что, при больших размерах, может быть очень затратно по времени.

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

При использовании линейного списка тратится больше времени на обработку операций с очередью, а так же может возникнуть фрагментация памяти. При реализации статическим кольцевым массивом, очередь всегда ограничена по размеру, но операции выполняются быстрее, нежели на списке.

8. **Что такое фрагментация памяти?**

Фрагментация – чередование занятых и свободных участков памяти при последовательных запросах на добавление и удаление. Свободные участки могут быть слишком малы, чтобы хранить в них нужную информацию.

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

Необходимо обратить внимание на корректное освобождение памяти при удалении элемента из очереди.

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

При запросе памяти, ОС находит подходящий блок памяти и записывает его в «таблицу» занятой памяти. При освобождении, ОС удаляет этот блок памяти из «таблицы» занятой пользователями памяти.