­­­­Лабораторная работа 4

Обработка очередей

Монахов Дмитрий ИУ7-34

Отчет (Вариант 13, задание 6)

**Условие задачи:**

### Реализовать операции работы со стеком, который представлен в виде массива (статического или динамического) и в виде односвязного линейного списка; оценить преимущества и недостатки каждой реализации: получить представление о механизмах выделения и освобождения памяти при работе со стеком.

### 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.

**Техническое задание**

**Аварийные ситуации**

Ошибка выделения памяти.

Запись (добавление элемента) в заполненную очередь.

Удаление из пустой очереди

Некорректная команда пользователя

**Предусмотрено:**

Сообщение об ошибке при выделении памяти.

Сообщение об ошибке записи.

Сообщение об ошибке удаления элемента.

Сообщение о некорректной команде.

**Входные данные:**

Команда пользователя.

Данные для заполнения очереди (ввод через консоль).

**Выходные данные:**

Сообщение об ошибке (если имеется).

Текущее состояние очередей.

Последний удаленный элемент.

Список адресов элементов очереди.

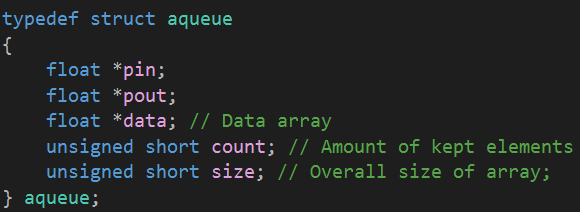
Список свободных областей.

Информация о времени выполнения операций добавления и удаления.

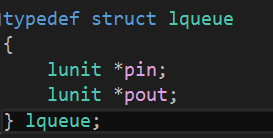
Информация о количестве используемой памяти.

**Структуры данных**

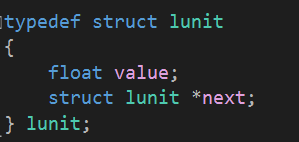
1. Очередь, основанная на динамическом массиве.



1. Очередь, основанная на списке



1. Элемент списка



**Функции**

Для очереди-массива:

int aqadd(aqueue \*queue, float x); // Добавление элемента в очередь

int aqremove(aqueue \*queue, float \*res); // Удаление элемента из очереди

int aqcreate(aqueue \*queue, int size); // Инициализация очереди

Для очереди-списка:

int lqadd(lqueue \*queue, float x); // Добавление элемента в очередь

int lqremove(lqueue \*queue, float \*result); // Удаление элемента из очереди

void lqcreate(lqueue \*queue); // Инициализация очереди

**Тесты**

1. Некорректная команда пользователя
2. Некорректный ввод данных для добавления
3. Добавление элемента в заполненный массив-очередь.
4. Удаление элемента из пустых очередей.

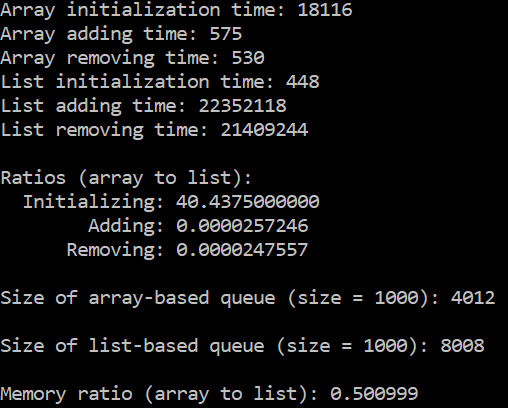
**Анализ**

**Сравнение видов реализации**

Первый вариант реализации – циклическая очередь на массиве с неизменяющимся размером, который можно задать при инициализации.

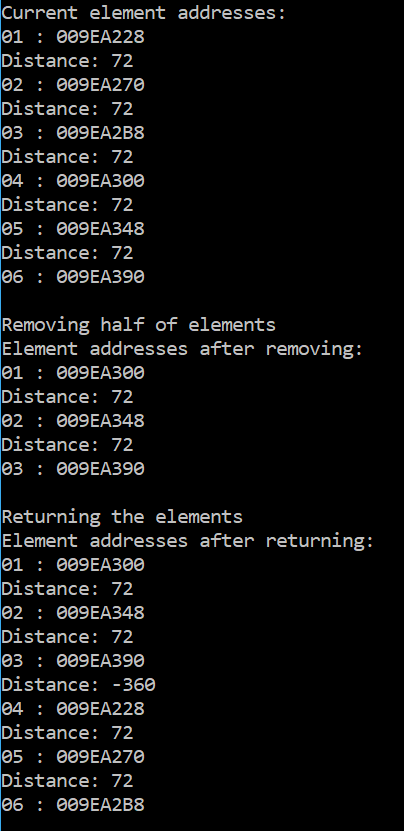
Второй вариант – очередь на списке.

Сравнение времени выполнения и занимаемой памяти:



Отсюда следует, что первый вариант реализации значительно эффективнее второго. Однако, очередь-список не подвержена такому недостатку как переполнение, в отличие от кольцевой очереди-массива.

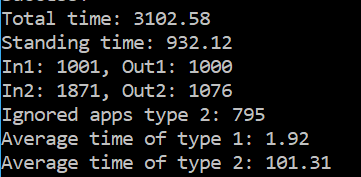
**Фрагментация памяти**

****

Из вывода программы вино, что расстояния между элементами в результате выполнения операций изменяются. Если расстояние между указателями на элементы равно 72, значит, они располагаются последовательно. В противном случае наблюдается фрагментация памяти.

**Оценка симуляции обрабатывающего автомата**

Вывод программы по окончанию симуляции:



Рассчитаем теоретическое время моделирования. Время обработки завки первого типа лежит в диапазоне от 0 до 4, значит среднее время – 2 е.в.

Среднее время прихода этой заявки – 3 е.в. Оно больше времени обработки, значит автомат будет работать с простоем. Общее время работы не зависит от обработки заявок второго типа, так как это автомат с абсолютным приоритетом для заявок первого типа. Для 1000 заявок суммарное время обработки составит 2 \* 1000 = 2000 е.в. Суммарное время прихода 1000 заявок равно 3 \* 1000 = 3000 е.в., этим числом и будет определяться время работы автомата. Время простоя вычисляется как разница между временем прихода и временем обработки 1000 заявок первого типа. Оно будет равно 3000 – 2000 = 1000 е.в.

Проверим правильность работы программы по выходу.

Найдем количество заявок, которое должно было обработаться за полученное на практике время: (3102 – 932) / 2 = 1085 заявок.

Вошло заявок 1076, тогда

|100%(1085 – 1076)/1076)| = 0.83%

Проверим по входу:

100%(3102 – 3000)/3000 = 3.4%

**Контрольные вопросы**

1. **Что такое очередь?**

Очередь – это последовательный список переменной длины, включение

Элементов в который идет с одной стороны (с хвоста), а исключение – с другой стороны (с головы). Работает по принципу FIFO.

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

При реализации в виде массива выделяется последовательная область памяти из m мест по L байт, где L – размер поля данных для одного элемента размещаемого типа.

При реализации в виде списка память выделяется отдельно для каждого элемента списка в размере M байт, где M складывается из размера типа данных, которые требуется хранить и указателя на следующий элемент списка.

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

В случае массива память освобождается только при удалении последнего элемента очереди. В случае списка память под очередной элемент освобождается при его удалении.

1. **Что происходит с элементами очереди при её просмотре?**

В случае реализации в виде одномерного массива, при удалении элемента необходимо сдвинуть все элементы очереди в сторону головы, что требует дополнительных временных затрат. Эта проблема решается с помощью использования кольцевой очереди или очереди-списка.

1. **Каким образом эффективнее реализовать очередь?**

См. Анализ, сравнение видов реализации

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

Очередь-массив быстрее и эффективнее по памяти. Однако ее недостаток в виде ограничения на объем не позволяет использовать этот вид реализации в случаях, когда количество данных заранее не известно или диапазон объемов слишком велик. В таком случае лучше будет реализовать очередь-список, которая не обладает этим недостатком.

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

Любые операции над очередью-массивом быстрее по времени и по памяти (см. Анализ, сравнение видов реализации). Однако во время инициализации выделяется большой блок памяти, поэтому инициализируется она медленнее, чем очередь-список. Так же при использовании линейного массива удаление элемента требует сдвига всех элементов в сторону головы, что снижает эффективность операции удаления. Это решается использованием кольцевой очереди, которая так же является массивом.

1. **Что такое фрагментация памяти?**

Фрагментация памяти это явление, при котором блоки занятой памяти не образуют непрерывную область. В результате этого появляются области различной длины, которые не могут быть заняты большими типами данных, но в сумме дают сильное поглощение свободного места.

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

На переполнение очереди-массива и на ошибки выделения памяти.

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

При выделении памяти программа ищет свободный непрерывный блок, помечает его как занятый и записывает в него данные. При невозможности найти такой блок (переполнение или фрагментация) программа может запросить у ОС дополнительную память.

При освобождении памяти программа помечает указанный блок как незанятый.