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 Городский Юрий Николаевич

Группа ИУ7 – 32Б

2023 г.

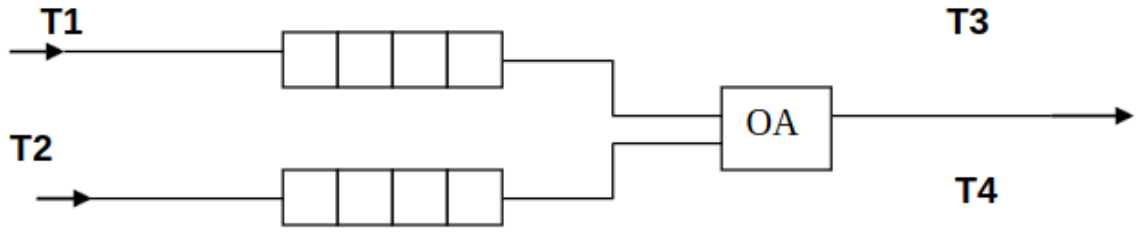
Оглавление

Условие задачи

Цель работы: отработка навыков работы с типом данных «очередь», представленным в виде одномерного статического массива (представление динамическим массивом можно добавить по желанию) 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.

Техническое задание

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:

1. **Номер команды:** целое число в диапазоне от 0 до 3

Выходные данные:

1. Состояние очереди-списка / очереди — статического массива при симуляции, использованные адреса.
    - Элементы в очереди 1 типа
    - Элементы в очереди 2 типа
    - Кол-во обработанных заявок 1 типа
    - Кол-во обработанных заявок 2 типа
    - Время обработки заявок 1 типа
    - Время обработки заявок 2 типа
    - Время простоя
    - Время работы аппарата
    - Прошедшее время с начала симуляции, отклонение от ожидаемого
    - Ожидаемое прошедшее время
    - Использованные адреса первой очереди
    - Была ли фрагментация в первой очереди
    - Использованные адреса второй очереди
    - Была ли фрагментация во второй очереди

2. Файлы с результатами временных замеров.

Функции программы

1. Симуляция на очереди - статическом массиве
2. Симуляция на очереди - списке
3. Замерный эксперимент
4. Выйти

Аварийные ситуации:

1. Некорректный ввод команды (введено не число или число не находится в диапазоне от 0 до 3): сообщение «Неверная команда».
2. Не удалось открыть файл при записи замерного эксперимента: сообщение «Ошибка работы с файлом».
3. Ошибка работы с памятью при симуляции: сообщение «Ошибка работы с динамической памятью»

Обращение к программе

Запуск через терминал (./app.exe).

Структуры данных

```
// Узел очереди - списка
struct queue_list_node
{
    void *data; // Данные
    queue_list_node_t *next; // Следующий узел
};
```

```
// Очередь - статический кольцевой массив
typedef struct
{
    object_t arr[QUEUE_ARRAY_EL_NUM]; // Массив элементов очереди
    int head; // Индекс первого элемента очереди
    int tail; // Индекс последнего элемента очереди
}queue_array_t;
```

```
// Симуляция очереди - статического массива
typedef struct
{
    queue_array_t *q_1;       // Очередь заявок 1 типа
    queue_array_t *q_2;       // Очередь заявок 2 типа
    size_t added_num_1;       // Количество добавленных заявок 1 типа
```

```
    size_t added_num_2;       // Количество добавленных заявок 2 типа
    size_t processed_num_1;   // Количество обработанных заявок 1 типа
    size_t processed_num_2;   // Количество обработанных заявок 2 типа
    double processed_time_1;  // Время обработки заявок 1 типа
    double processed_time_2;  // Время обработки заявок 2 типа
    double break_time;        // Время простоя
    double summ_process_time; // Время работы обрабатывающего аппарата без
простоя
    double elapsed_time;      // Время, прошедшее с момента поступления первого
элемента
    double expected_time;     // Расчетное время работы аппарата
    int last_processed_type;  // Тип последней обработанной заявки
    address_node_t *used_addresses_1; // Использованные адреса 1 очереди
    address_node_t *used_addresses_2; // Использованные адреса 2 очереди
} process_array_t;
```

```
// Симулияция очереди - списка
typedef struct
{
    queue_list_node_t *q_1;   // Очередь заявок 1 типа
    queue_list_node_t *q_2;   // Очередь заявок 2 типа
    size_t added_num_1;       // Количество добавленных заявок 1 типа
    size_t added_num_2;       // Количество добавленных заявок 2 типа
    size_t processed_num_1;   // Количество обработанных заявок 1 типа
    size_t processed_num_2;   // Количество обработанных заявок 2 типа
    double processed_time_1;  // Время обработки заявок 1 типа
    double processed_time_2;  // Время обработки заявок 2 типа
    double break_time;        // Время простоя
    double summ_process_time; // Время работы обрабатывающего аппарата без
простоя
    double elapsed_time;      // Время, прошедшее с момента поступленяи первого
элемнта
    double expected_time;     // Расчетное время работы аппарата
    int last_processed_type;  // Тип последней обработанной заявки
    address_node_t *used_addresses_1; // Использованные адреса 1 очереди
    address_node_t *used_addresses_2; // Использованные адреса 2 очереди
} process_t;
```

Тесты

Таблица 1: Негативные тесты

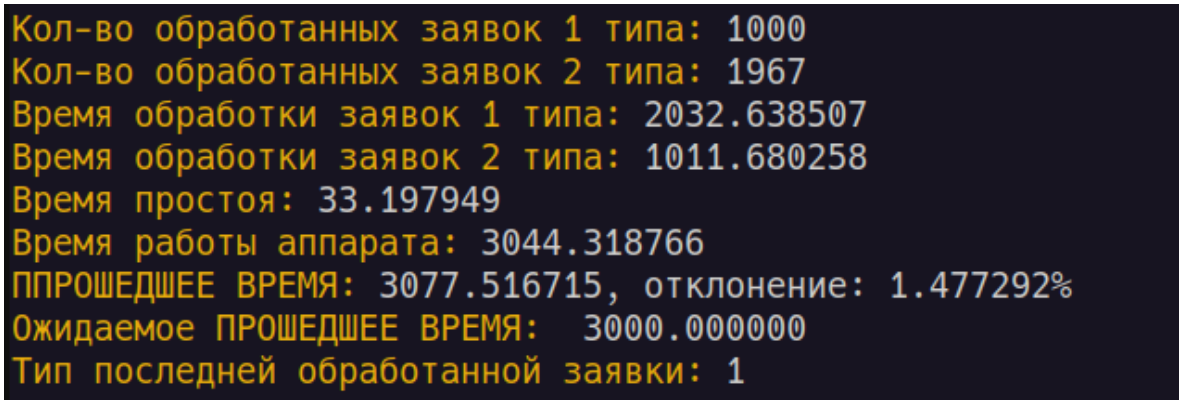
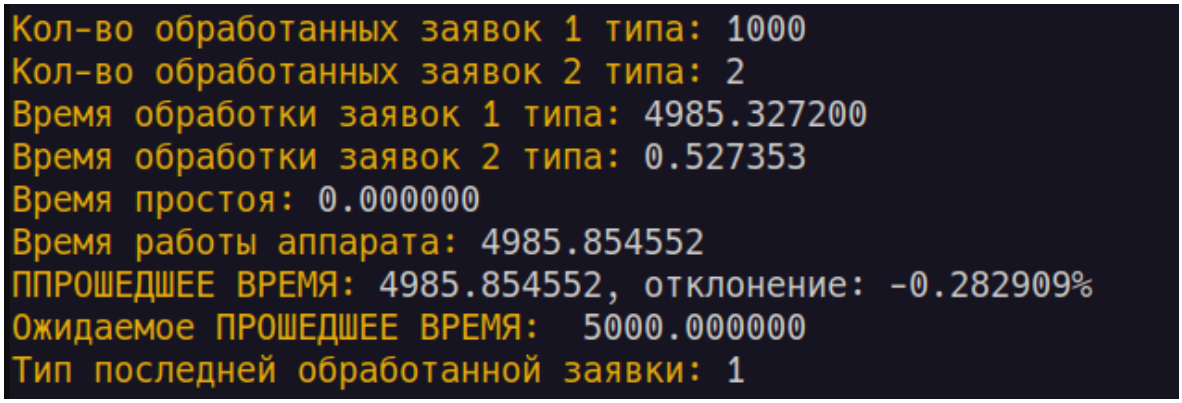
Описание	Ввод	Вывод
Текст вместо команды	abc	Неверная команда
Неверный диапазон команды (> 3)	4	Неверная команда
Неверный диапазон команды (< 0)	-1	Неверная команда

Таблица 2: Пример симуляции на очереди — статическом массиве

№	Выход
1	**Расчетные данные** Время поступление в первую очередь [1,5] Время поступление в вторую очередь [0,3] Время обработки из первой очереди [0,4] Время обработки из второй очереди [0,1] **Ожидаемый результат:** Время добавления в первую очередь дольше времени обработки,следовательно Время моделирования = среднее время прихода в первую очередь * 1000 = = 3 * 1000 = 3000
	Кол-во обработанных заявок 1 типа: 1000 Кол-во обработанных заявок 2 типа: 1951 Время обработки заявок 1 типа: 2012.653396 Время обработки заявок 2 типа: 968.953619 Время простоя: 13.947636 Время работы аппарата: 2981.607015 ППРОШЕДШЕЕ ВРЕМЯ: 2995.554651, отклонение: -0.613099% Ожидаемое ПРОШЕДШЕЕ ВРЕМЯ: 3000.000000 Тип последней обработанной заявки: 1
	Расчетные данные Время поступление в первую очередь [1,5] Время поступление в вторую очередь [0,3] Время обработки из первой очереди [0,10] Время обработки из второй очереди [0,1] **Ожидаемый результат:** Время обработки из первой очереди дольше времени добавления,следовательно Время моделирования = среднее время обработки из первой очереди * 1000 = = 5 * 1000 = 5000
	Кол-во обработанных заявок 1 типа: 1000 Кол-во обработанных заявок 2 типа: 9 Время обработки заявок 1 типа: 5046.267692 Время обработки заявок 2 типа: 5.655765 Время простоя: 0.636842 Время работы аппарата: 5051.923457 ППРОШЕДШЕЕ ВРЕМЯ: 5052.560299, отклонение: 1.038469% Ожидаемое ПРОШЕДШЕЕ ВРЕМЯ: 5000.000000 Тип последней обработанной заявки: 1

Таблица 3: Пример симуляции на очереди — списке

№	Выход
1	**Расчетные данные** Время поступление в первую очередь [1,5] Время поступление в вторую очередь [0,3]

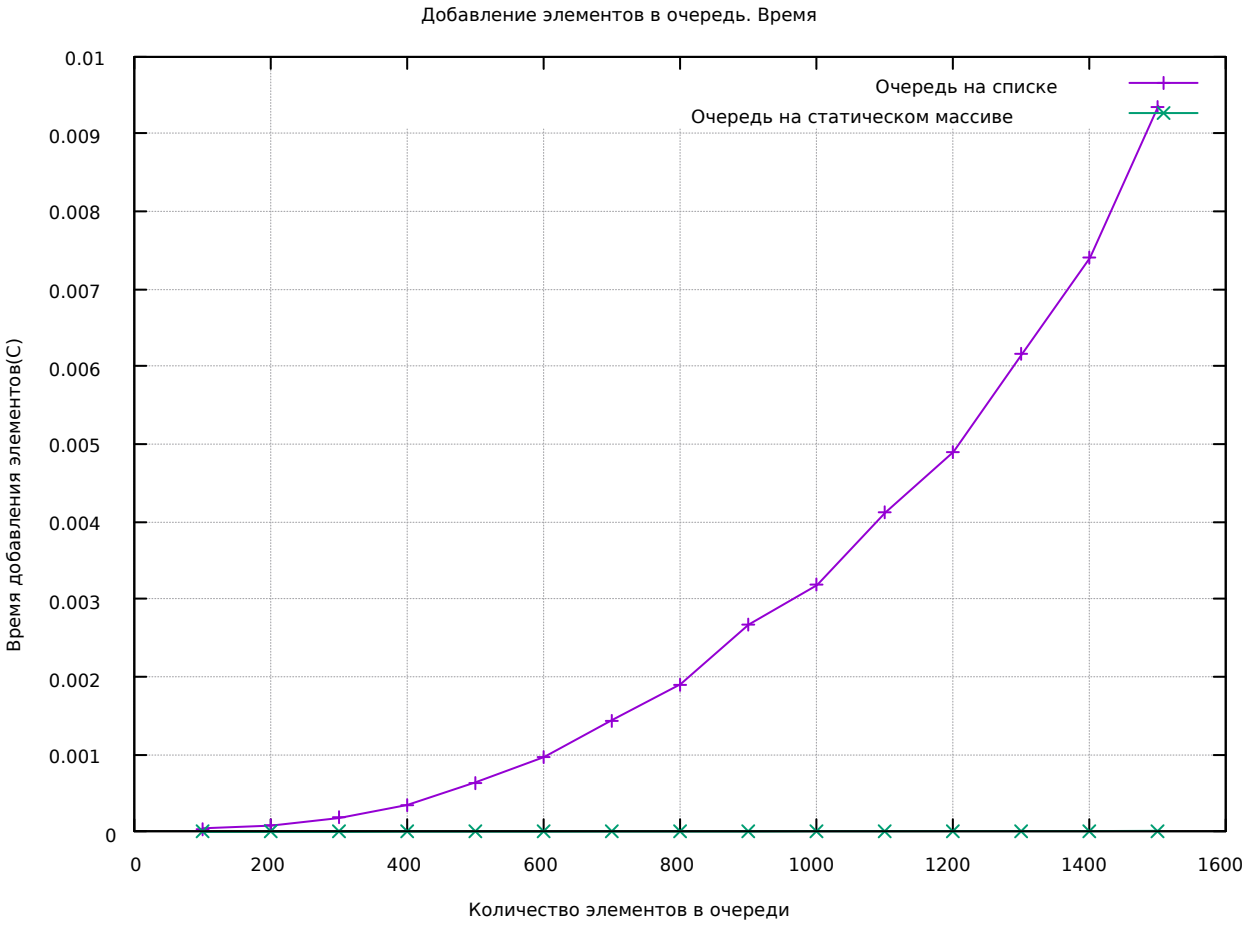
	Время обработки из первой очереди [0,4] Время обработки из второй очереди [0,1] **Ожидаемый результат:** Время добавления в первую очередь дольше времени обработки,следовательно Время моделирования = среднее время прихода в первую очередь * 1000 = = 3 * 1000 = 3000
	
2	**Расчетные данные** Время поступление в первую очередь [1,5] Время поступление в вторую очередь [0,3] Время обработки из первой очереди [0,10] Время обработки из второй очереди [0,1] **Ожидаемый результат:** Время обработки из первой очереди дольше времени добавления,следовательно Время моделирования = среднее время обработки из первой очереди * 1000 = = 5 * 1000 = 5000
	

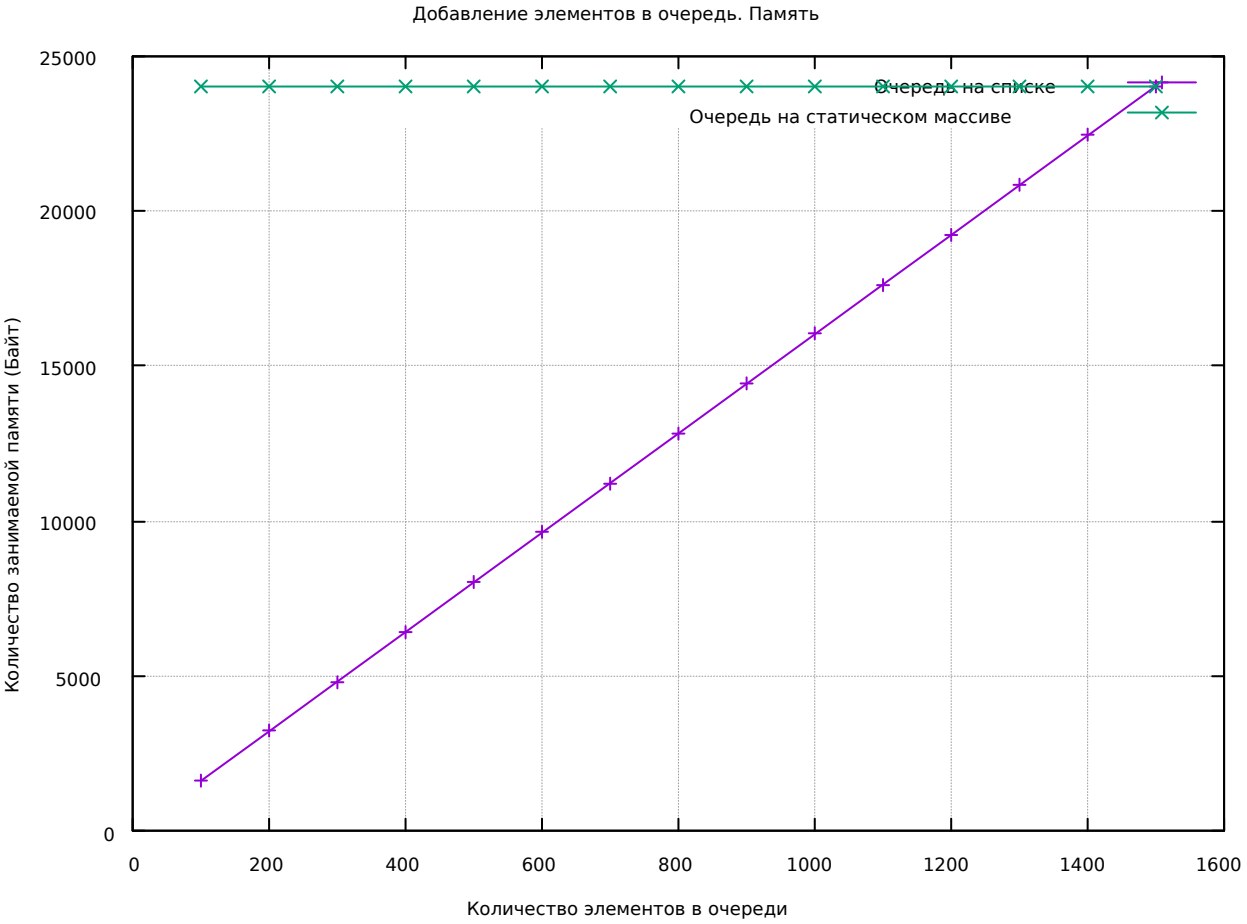
Фрагментация

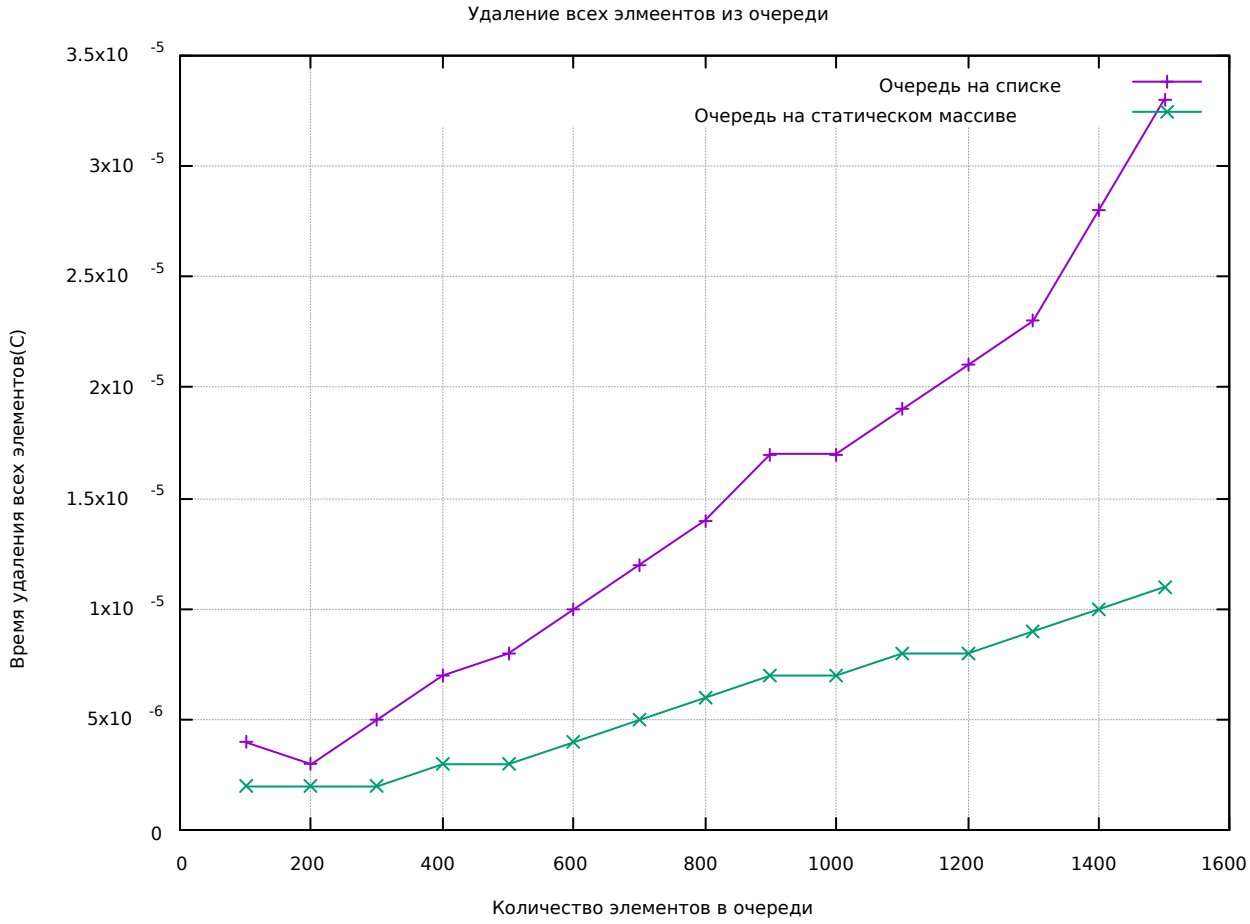
Очередь на статическом массиве	Очередь на списке
Кол-во обработанных заявок 1 типа: 1000 Кол-во обработанных заявок 2 типа: 1995 Время обработки заявок 1 типа: 1980.015609 Время обработки заявок 2 типа: 1003.896179 Время простоя: 35.198351 Время работы аппарата: 2983.911788 ППРОШЕДШЕЕ ВРЕМЯ: 3019.110139, отклонение: -0.536274% Ожидаемое ПРОШЕДШЕЕ ВРЕМЯ: 3000.000000 Тип последней обработанной заявки: 1 Количество различных адресов: 47 0x56482a408b40 251 0x56482a408b50 84 0x56482a408b60 67 0x56482a408b70 53 0x56482a408b80 52 0x56482a408b90 47 0x56482a408ba0 44 0x56482a408bb0 39 0x56482a408bc0 36 0x56482a408bd0 34 0x56482a408be0 31 0x56482a408bf0 29 0x56482a408c00 25 0x56482a408c10 23 0x56482a408c20 21 0x56482a408c30 21 0x56482a408c40 19 0x56482a408c50 13 0x56482a408c60 12 0x56482a408c70 11 0x56482a408c80 11 0x56482a408c90 8 0x56482a408ca0 7 0x56482a408cb0 6 0x56482a408cc0 5 ... Фрагментации не было	Кол-во обработанных заявок 1 типа: 1000 Кол-во обработанных заявок 2 типа: 2083 Время обработки заявок 1 типа: 2006.465569 Время обработки заявок 2 типа: 1035.925951 Время простоя: 30.953448 Время работы аппарата: 3042.391520 ППРОШЕДШЕЕ ВРЕМЯ: 3073.344969, отклонение: 1.413051% Ожидаемое ПРОШЕДШЕЕ ВРЕМЯ: 3000.000000 Тип последней обработанной заявки: 1 Количество различных адресов: 657 0x55d90e3ec640 1 0x55d90e3f89c0 1 0x55d90e381f40 1 0x55d90e3f8a20 1 0x55d90e3f8a60 1 0x55d90e3f8ba0 1 0x55d90e3f8ce0 1 0x55d90e3f8d40 1 0x55d90e3f8cc0 1 0x55d90e3f8f20 1 0x55d90e3f8b00 2 0x55d90e3f8d80 1 0x55d90e3f8f60 1 0x55d90e3f8c00 1 0x55d90e3f8e20 1 0x55d90e3f8b40 1 0x55d90e3f8c60 1 0x55d90e3f8ec0 1 0x55d90e3f8de0 1 0x55d90e3f8ee0 1 0x55d90e3f8ca0 1 0x55d90e3f8b20 1 0x55d90e3f8fe0 1 0x55d90e3f8b80 1 0x55d90e3f90a0 1 ... Фрагментация была

Замерный эксперимент

Замеры проводились на запись и удаление элементов в очередь — список и очередь — статический массив. При каждом замере делалось 1000 повторов, среднее значение записывалось в файл, по которому построены графики.




Добавление элементов в очередь. Память
Очередь на списке
Очередь на статическом массиве
Количество занимаемой памяти (Байт)
Количество элементов в очереди
0
5000
10000
15000
20000
25000
0
200
400
600
800
1000
1200
1400
1600


Удаление всех элмеентов из очереди
Очередь на списке
Очередь на статическом массиве
Время удаления всех элементов(С)
Количество элементов в очереди
0
5x10 -6
1x10 -5
1.5x10 -5
2x10 -5
2.5x10 -5
3x10 -5
3.5x10 -5
0
200
400
600
800
1000
1200
1400
1600

Контрольные вопросы

1. Что такое FIFO и LIFO?

Способы организации данных. Первым пришел — первым вышел (First In – First Out (FIFO)) – так реализуются очереди. Первым пришел — последним вышел, т. е. Last In – First Out (LIFO)- так реализуются стеки.

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

При реализации массивом единожды выделяется память для хранения только самих элементов. При реализации в виде списка для каждого элемента дополнительно выделяется память для хранения адреса следующего элемента.

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

При реализации очереди списком указателю голове списка присваивается значение следующего элемента списка, а память из-под удаляемого элемента освобождается.

При реализации очереди массивом память из-под удаляемого элемента не освобождается. Освобождение будет происходить аналогично – единожды для всего массива.

4. Что происходит с элементами очереди при ее просмотре?

При просмотре очереди элементы удаляются и происходит очистка очереди, влекущая за собой освобождение памяти в случае реализации ее списком

5. От чего зависит эффективность физической реализации очереди?

Эффективнее и по памяти, и по времени реализовывать очередь массивом. Это зависит от самой структуры элемента в случае памяти, и от запросов в оперативную память в случае времени.

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

В случае массива недостатком является фиксированный размер очереди. В случае односвязного списка недостатками являются затраты по скорости и фрагментация.

7. Что такое фрагментация памяти?

Фрагментация — возникновение участков памяти, которые не могут быть использованы. Фрагментация может быть внутренней — при выделении памяти блоками остается не задействованная часть, может быть внешней — свободный блок, слишком малый для удовлетворения запроса

8. Для чего нужен алгоритм «близнецов»?

Идея этого алгоритма состоит в том, что организуются списки свободных блоков отдельно для каждого размера 2^k, $0 <= k <= m$. Вся область памяти кучи состоит из 2^m элементов, которые, можно считать, имеют адреса с 0 по $2^m - 1$. Первоначально свободным является весь блок из 2^m элементов. Далее, когда требуется блок из 2^k элементов, а свободных блоков такого размера нет, расщепляется на две равные части блок большего размера; в результате появится блок размера 2^k (т.е. все блоки имеют длину, кратную 2). Когда один блок расщепляется на два (каждый из которых равен половине первоначального), эти два блока называются ***близнецами***. Позднее, когда оба близнеца освобождаются, они опять объединяются в один блок.

9. Какие дисциплины выделения памяти вы знаете?

Две основные дисциплины сводятся к принципам "самый подходящий" и "первый подходящий". По дисциплине "самый подходящий" выделяется тот свободный участок, размер которого равен запрошенному или превышает его на минимальную величину. По дисциплине "первый подходящий" выделяется первый же найденный свободный участок, размер которого не меньше запрошенного.

10. На что необходимо обратить внимание при тестировании программы?

При тестировании программы необходимо обратить внимание на корректность выводимых данных, проследить за выделением и освобождением выделяемой динамически памяти, предотвратить возможные аварийные ситуации.

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

В оперативную память поступает запрос, содержащий необходимый размер выделяемой памяти. Выше нижней границы свободной кучи осуществляется поиск блока памяти подходящего размера. В случае если такой найден, в вызываемую функцию возвращается указатель на эту область и внутри кучи она помечается как занятая. Если же найдена область, большая необходимого размера, то блок делится на две части, указатель на одну возвращается в вызываемую функцию и помечается как занятый, указатель на другую остается в списке свободных областей. В случае если области памяти необходимого размера не было найдено, в функцию возвращается NULL. При освобождении памяти происходит обратный процесс. Указатель на освобождаемую область поступает в оперативную память, если это возможно объединяется с соседними свободными блоками, и помечается свободными.

Вывод

Очередь основанная на статическом массиве работает быстрее и использует меньше памяти чем очередь на односвязном списке. Но если надо будет хранить больше элементов, чем размер статического массива, то в очереди на статическом массиве этого сделать не получится. Поэтому для очередей с неизвестным размером лучше использовать очереди на списке.