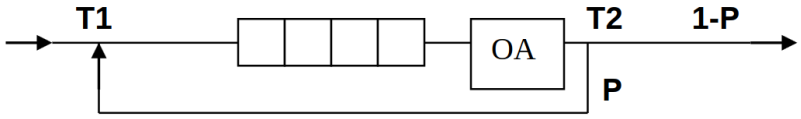
Условие задачи:

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и очереди заявок.



Заявки поступают в "хвост" очереди по случайному закону с интервалом времени T1, равномерно распределенным от 0 до 6 единиц времени (е.в.). В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время T2 от 0 до 1 е.в., Каждая заявка после ОА с вероятностью P=0.8 вновь поступает в "хвост" очереди, совершая новый цикл обслуживания, а с вероятностью 1-P покидает систему. (Все времена – вещественного типа). В начале процесса в системе заявок нет.

Смоделировать процесс обслуживания до ухода из системы первых 1000 заявок. Выдавать после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди. В конце процесса выдать общее время моделирования и количество вошедших в систему и вышедших из нее заявок, среднее время пребывания заявки в очереди, время простоя аппарата, количество срабатываний ОА.

Обеспечить по требованию пользователя выдачу
на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Техническое задание:

1. Входные данные:

Целые числа от 0 до 5.

2. Выходные данные:

В зависимости от введенной команды:

0. Сообщение “Программа искусственно завершена.”.
1. Реализация очереди на массиве до 1000 обработанных элементов.
2. Реализация очереди на списке до 1000 обработанных элементов.

3. Вывод задействованных адресов (для списка), с обозначениями, обработан уже узел с этим адресом или нет. (при выводе этого пункта мы видим что при реализации очереди списком происходит фрагментация памяти)

4. Сравнение работы очереди на списке и на массиве.

5. Вывод для 4 различных размерностей очереди.

Аварийные ситуации:

1. Некорректный ввод команды.

Теоретические расчеты:

Время моделирования – максимальное из среднего времени прихода заявок и среднего времени обработки заявок умножить на количество заявок

Ожидаемое время обработки – среднее время обработки умноженное на количество заявок и на 1/(1 - P)

Время простоя аппарата – время моделирования – ожидаемое время обработки

Количество срабатывания аппарата – 1 /(1 - P) умножить на количество заявок

По заданию:
Время поступления – от 0 до 6 единиц времени
Время обслуживания – от 0 до 1 единицы времени
Вероятность возвращения в “хвост” – 0.8

Расчеты:
Количество вошедших заявок – 1000
Количество вышедших заявок – 1000
Время простоя аппарата – 500
Количество срабатываний ОА - 5000

Выдача программы:

```
Ожидаемое время моделирования: 3000.00
Полученное время моделирования: 3067.53
Погрешность: ~2.25%

Количество вошедших заявок: 1004
Количество вышедших заявок: 1000
Среднее время в очереди: 6.92
Время простоя аппарата: 530.08
Количество срабатывания аппарата: 5070

Время для списка(в тиках): 3394606
Время для массива(в тиках): 2975658
Память для списка(в байтах): 208
Память для массива(в байтах): 8000
```

Описание структур данных:

Очередь в виде списка:

struct queue_slot *arr = malloc(all * sizeof(struct queue_slot));

```
struct queue_slot
{
double arrival_time; //время прихода в очередь
struct queue_slot *next; //указатель на след элемент
};
```

```
struct queue_slot     arr = malloc(all      sizeof(struct queue_slot));
```

Массив таких структур, в next ничего не пишется. (очередь в виде массива)

Очередь:

```
struct queue
{
struct queue_slot *pin; //указатель на начало очереди
struct queue_slot *pout; //указатель на конец очереди
int len; //длина очереди
int in_num; //число вошедших в очередь заявок
long long int state; //переменная для вычисления средней длины
int max; // переменная для подсчета очереди в списке
double total_stay_time; //время нахождения заявок в очереди
};
```

Обслуживающий аппарат(ОА):

```
struct machine
{
double time; //текущее время состояния аппарата
double downtime; // время простоя аппарата
int triggering; // количесво срабатывания аппарата
int processed_count; //кол-во обработанных из очереди заявок
};
```

Массив адресов:

```
struct mem_slot
{
struct queue_slot *queue_slot; //указатель на участок памяти
int busy; // состояние участка 1(занят) или 0
struct mem_slot *next; //указатель на след элемент очереди
};
```

Тесты:

Входные данные	Вывод программы	Код возврата
0	Программа искусственно завершена	0
1 (очередь из 1000 элементов в виде списка)	Каждые 100 обработанных элементов выводит данные: количество заявок, текущая длина очереди, средняя длина очереди. После обработки всей очереди информация: ожидаемое время моделирования, реальное время моделирования, погрешность, количество вошедших заявок, количества вышедших заявок, среднее время в очереди, время простоя аппарата, количество срабатывания аппарата.	Программа делает запрос на ввод следующей команду.
2 (очередь из 1000 элементов в виде списка)	Каждые 100 обработанных элементов выводит данные:	Программа делает запрос на ввод следующей команду.

	количество заявок, текущая длина очереди, средняя длина очереди. После обработки всей очереди информация: ожидаемое время моделирования, реальное время моделирования, погрешность, количество вошедших заявок, количества вышедших заявок, среднее время в очереди, время простоя аппарата, количество срабатывания аппарата.	
3 (вывод таблицы использованных адресов) если до этого вызывался пункт 2.	Вывод адресов с дополнительными символами: “-” в случае если адрес уже не используется, “+”, если адрес задействован сейчас. При вызове этой команды список использованных адресов очищается.	Программа делает запрос на ввод следующей команду.
3 (вывод таблицы использованных адресов) если до этого не вызывался пункт 2.	Сообщение “Для данного пункта нужно смоделировать очередь списком.”	Программа делает запрос на ввод следующей команду.
4 (сравнение работы очереди на массиве и на списке)	Запуск команд 1 и 2, соответствующий вывод, далее выводятся данные для сравнения количества памяти и времени, используемой для реализации очереди на массиве и списке.	Программа делает запрос на ввод следующей команду.
5 (вывод результатов по очередям различных	Вывод 1 и 2 пунктов для размерностей 100, 500,	Программа делает запрос на ввод следующей

размерностей)	1000, 2000.	команду.
Попытка ввода некорректного номера пункта из меню.	Сообщение “Введён неверный ключ.”	1

Замеры эффективности программы:

Количество запросов, вышедших из ОА	Время для списка (в тиках)	Время для массива (в тиках)	Память, задействованная для списка (в байтах)	Память, задействованная для массива (в байтах)
100	407586	318698	192	800
500	1692946	1412078	240	4000
1000	3272116	2860378	208	8000
2000	6171042	5178836	288	16000

Вывод: исходя из таблицы очередь на списке работает медленнее, чем очередь на массиве. Но сильно выигрывает по памяти, так как освобожденные узлы списка больше не задействуют память, а массив так и остается с тем же размером. Таким образом, когда при использовании очереди надо задействовать меньше времени, стоит использовать массив, а когда нам важнее сэкономить память, стоит использовать список.

Сравнение реализации FIFO и LIFO:

Как в этой, так и в прошлой лабораторной работе, мы выяснили, что минус реализации стека/очереди через массив (когда мы не знаем количество элементов очереди/стека) – большое количество лишней памяти, которая задействована во время выполнения программы. Но их реализация на массиве обрабатывается быстрее, чем их реализация на списке. В свою очередь реализация стека/очереди на списке тратит сильно меньше памяти(когда мы не знаем количество элементов очереди/стека), но время обработки – слабая черта этого метода хранения очереди/стека так как, его тратится больше чем, на реализацию посредством массива. Если заранее известно количество элементов очереди, удобнее пользоваться массивом, так как в таком случае мы не будем выделять лишнюю память, таким образом и память лишняя не будет выделяться и скорость выполнения будет меньше.

Кол-во элементов	Заполнение стека на массиве, мкс	Заполнение стека на списке, мкс	Заполнение очереди на массиве, мкс	Заполнение очереди на списке, мкс
1	0.3	0.6	1	0.8
10	30	80	29	76
1000	3140	7598	3080	7609
10000	31035	76999	36758	70908
100000	309987	771990	325465	757460

Таким образом стек и очередь заполняются примерно одинаковое количество времени. Потому что мы делаем одно и то же коичество действий для добавления элемента в очередь и в стек.

Эти типы данных различаются тем, что используются в разных алгоритмах.
В тех, где нам нужен доступ к последнему добавленному элементу, мы используем стек, там где нам нужен доступ к элементу, который был добавлен первым, мы используем очередь.

Контрольные вопросы:

1. Что такое FIFO и LIFO?

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

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 pout присваивается значение следующего элемента списка, а память из-под удаляемого элемента освобождается.

При реализации очереди массивом память из-под удаляемого элемента не освобождается, так как изначально она была выделена под весь массив. 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?

Выяснилось, что 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.