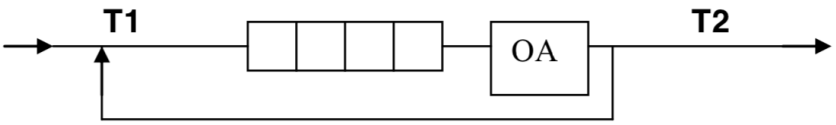
Типы и структуры данных — Сиденко Анастасия
Лабораторная работа 5
Вариант 7 — ИУ7-33Б
Обработка очередей

Техническое задание:

Задача:

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 вновь поступает в "хвост" очереди, совершая всего 5 циклов обслуживания, после чего покидает систему. (Все времена – **вещественного** типа) В начале процесса в системе заявок нет.

Смоделировать процесс обслуживания до ухода из системы первых 1000 заявок, выдавая после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди, а в конце процесса - общее время моделирования и количестве вошедших в систему и вышедших из нее заявок, количестве срабатываний ОА, время простоя аппарата. По требованию пользователя выдать на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Входные данные: время обработки заявок и прихода в очередь.

Выходные данные: время моделирования, средняя очередь.

Взаимодействие с программой

Взаимодействие через консольное меню.

```
Как будем хранить очередь?
1. Массивом
2. Списком
2
Введите границу времени для поступления заявок
6
Введите границу времени для работы аппарата
1
```

Аварийные ситуации:

1. Некорректные данные. Вывод сообщения.

Используемые структуры:

Структура список

```
struct list {
    struct list *next;
    int *data;
};
```

Массив свободных областей памяти.

```
struct list *massiv_free[10000] = {0};
int free_kol = 0;
```

Очередь в виде массива

```
struct queue_massiv {
  int *qu[QMAX];
  int pin, pout;
};
```

Очередь в виде списка

```
struct queue_list {
    int size;
    struct list *pout, *pin;
};
```

Алгоритмы:

1. Инициализация очереди в виде массива
   void init_massiv(struct queue_massiv *q)

2. Добавление элемента в очередь в виде массива
   void push_massiv(struct queue_massiv *q, int *x)

3. Проверка на пустоту очереди в виде массива
   int isempty_massiv(struct queue_massiv *q)

4. Удаление элемента из очереди в виде массива
   int *pop_massiv(struct queue_massiv *q)

5. Инициализация очереди в виде списка
   struct queue_list *init_list(void)

6. Добавление элемента в очередь в виде списка
   void push_list(struct queue_list *lt, int *data)

7. Удаление элемента из очереди в виде списка
   int *pop_list(struct queue_list *lt)

Тесты:

1. Некорректные данные

Ввод некорректных данных

```
Как будем хранить очередь?
1. Массивом
2. Списком
q
Некорректное значение
```

```
Как будем хранить очередь?
1. Массивом
2. Списком
1
Введите границу времени для поступления заявок
q
Некорректное значение!
```

2. Правильные тесты

```
Очередь 0
Средняя очередь 0
Общее время моделирования 2952.565474
Количество вышедших заявок 1000
Количество пришедших заявок 1000
Количество срабатываний аппарата 5000
Время простоя 429.213867
```

```
Средняя очередь 0
Очередь 1
Средняя очередь 0
Очередь 0
Средняя очередь 0
Общее время моделирования 3044.360156
Количество вышедших заявок 1000
Количество пришедших заявок 1000
Количество срабатываний аппарата 5000
Время простоя 569.202148
Количество ячеек памяти, которые были освобождены, но не заняты 10
Вывести?
 1 - да
 2 - нет
1
0x7f95314015b0 0x7f95314015c0 0x7f9532800000 0x7f9532a00000 0x7f95314015d0
```

Сравнение времени работы и используемой памяти.

При хранении в виде массива очередь ограничена 10000 элементами, а значит память необходимая для хранения такой очереди 800016 байт.

При хранении в виде списка очередь ограничена только объемом доступной оперативной памяти.

Время работы очереди в виде массива меньше всего на 25%.

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»). Принцип работы очереди: первым пришел – первым вышел, т. е. First In – First Out (FIFO).

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

При моделировании линейной очереди на основе одномерного массива выделяется последовательная область памяти из m мест по L байт, где L – размер поля данных для одного элемента размещаемого типа.

При моделировании линейной очереди на основе односвязного списка в статической памяти храниться адрес начала и конца очереди.

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

При реализации одномерным массивом очередь освобождается после выполнения программы.

При использовании односвязного спила память освобождается сразу после удаления из очереди.

4. Что происходит с элементами очереди при ее просмотре?

При выходе из очереди (удаление из начала) элемент добавляется в ее конец.

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

   Если заранее известен размер очереди, эффективнее использовать массив, так как в списке кроме данных хранится еще и указатель.
   Если размер неизвестен, то массив выделяется с запасом, а значит список будет эффективнее по памяти так как там память выделяется динамически.
   Если нам нужна эффективность по времени, то однозначно нужно использовать массив.

6. Что такое фрагментация памяти?

Участки свободной и занятой памяти могут чередоваться.

7. На что необходимо обратить внимание при тестировании программы?

На корректность данных и переполнение очереди при использовании статического массива.

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

В памяти находится необходимая нам для выделения область, она резервируется по данные, при освобождении память освобождается и ее можно использовать в других местах.

Вывод:

Если заранее известен размер очереди, эффективнее использовать массив, так как в списке кроме данных хранится еще и указатель.
Если размер неизвестен, то массив выделяется с запасом, а значит список будет эффективнее по памяти так как там память выделяется динамически.
Если нам нужна эффективность по времени, то однозначно нужно использовать массив.