

# Семинар #11: Динамический массив. Стек и Очередь. Домашнее задание.

## Очередь

```
#define CAPACITY 7
typedef int Data;

struct queue
{
    int front;
    int back;
    Data values[CAPACITY];
};
typedef struct queue Queue;

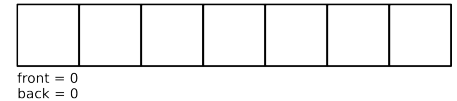
// .....

int main()
{
    Queue a;
    queue_init(&a);
    enqueue(&a, 100);
    for (int i = 0; i < 20; ++i)
    {
        enqueue(&a, i);
        dequeue(&a);
    }
    enqueue(&a, 200);
    queue_print(&a);
}
```

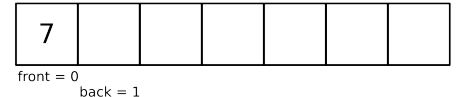
Очередь — абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел». Реализация с помощью массива:

*Queue b;*  
*b.front = 0;*  
*b.back = 0;*

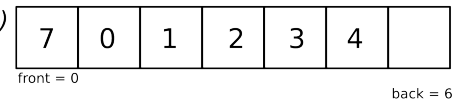
values:



*enqueue(&b, 7);*



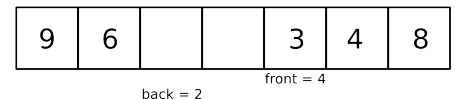
*for (int i = 0; i < 5; ++i)*  
*enqueue(&b, i);*



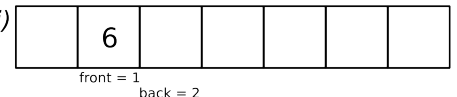
*for (int i = 0; i < 4; ++i)*  
*dequeue(&b);*



*enqueue(&b, 8);*  
*enqueue(&b, 9);*  
*enqueue(&b, 6);*



*for (int i = 0; i < 4; ++i)*  
*dequeue(&b);*



### Задача #3: Очередь на основе статического массива:

1. Написать функцию `void queue_init(Queue* q)`, которая будет задавать начальные значения полей `front` и `back`.
2. Написать функцию `void enqueue(Queue* q, Data x)` - добавляет `x` в очередь. Для эффективной реализации очереди, нужно использовать как можно меньше операций и как можно эффективней использовать выделенную память. Поэтому, при заполнении массива, если начало массива свободно, то элементы можно хранить там. (смотрите рисунок)
3. Написать функцию `Data dequeue(Queue* q)` - удаляет элемент из очереди и возвращает его. Для эффективной реализации очереди сдвигать оставшиеся элементы не нужно. Вместо этого можно просто увеличить поле `front`.
4. Написать функцию `int queue_is_empty(const Queue* q)`, которая возвращает 1 если очередь пуста и 0 иначе.
5. Написать функцию `int queue_get_size(const Queue* q)`, которая возвращает количество элементов.

6. Написать функцию `int queue_is_full(const Queue* q)`, которая возвращает 1 если очередь заполнена и 0 иначе. Очередь считается полной, если `size == capacity - 1`.
7. Написать функции `Data queue_get_front(const Queue* q)` и `Data queue_get_back(const Queue* q)`, которые возвращают элементы, находящиеся в начале и в конце очереди соответственно, но не изменяют очередь.
8. Написать функцию `void queue_print(const Queue* q)`, которая распечатывает все элементы очереди.
9. Что произойдёт, если вызвать `enqueue` при полной очереди или `dequeue` при пустой? Обработайте эти ситуации. Программа должна печатать сообщение об ошибке и завершаться с аварийным кодом завершения. Чтобы завершить программу таким образом можно использовать функцию `exit` из библиотеки `stdlib.h`.
10. Протестируйте очередь на следующих тестах:
  - (a) В очередь добавляется 4 элемента, затем удаляется 2. Вывести содержимое очереди с помощью `queue_print()`
  - (b) В очередь добавляется очень много элементов (больше чем `CAPACITY`). Программа должна напечатать сообщение об ошибке.
  - (c) В очередь добавляется 3 элемента, затем удаляется 2, затем добавляется очень много элементов (больше чем `CAPACITY`). Программа должна напечатать сообщение об ошибке.
  - (d) В очередь добавляется 3 элемента, затем удаляется 4. Программа должна напечатать сообщение об ошибке.
  - (e) В очередь добавляется 2 элемента, затем выполняется следующий цикл:

```
for (int i = 0; i < 10000; ++i)
{
    enqueue(&a, i);
    dequeue(&a);
}
```

Вывести содержимое очереди с помощью `queue_print()`

#### Задача #4: Очередь на основе динамического массива:

Описание такой очереди выглядит следующим образом:

```
struct queue
{
    int capacity;
    int front;
    int back;
    Data* values;
};
typedef struct queue Queue;
```

1. Скопируйте код очереди со статическим массивом в новый файл и измените описание структуры как показано выше. Макрос `CAPACITY` больше не нужен, его можно удалить.
2. Измените функцию `void queue_init(Queue* q)` на `void queue_init(Queue* q, int initial_capacity)`. Теперь она должна присваивать `capacity` начальное значение `initial_capacity` и выделять необходимую память под массив `values`.
3. Измените функцию `void enqueue(Queue* q)`. Теперь, при заполнении очереди должно происходить пере-выделение памяти с помощью функции `realloc`. Заполнение очереди достигается когда размер очереди становится равным `capacity - 1` (а не `capacity`, потому что при полном заполнении вместимости `front` будет равняться `back` и мы не сможем понять полная эта очередь или пустая). После перевыделения нужно переместить элементы массива на новые места и изменить `front` и `back`. Если `front != 0`, то нужно переместить элементы массива от `front` до конца старого массива `values` в конец нового массива `values`. (смотрите рисунок ниже)
4. Добавьте функцию `void queue_destroy(Queue* q)`, которая будет освобождать память, выделенную под массив `values`.

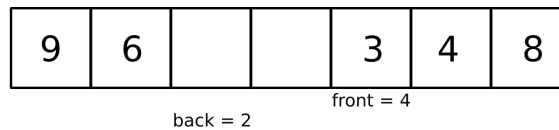
5. Протестируйте очередь: в очередь добавляется много элементов ( $\gg 10^3 > \text{initial\_capacity}$ ). Программа **не** должна напечатать сообщение об ошибке (если только совокупный размер элементов не превышает размер доступной оперативной памяти).
6. В случае, если `malloc` или `realloc` не смогли выделить запрашиваемый объём памяти (например, по причине того, что этот объём больше, чем вся доступная оперативная память или по какой-нибудь иной причине), то они возвращают значение `NULL`. Программа должна это учитывать и завершаться с ошибкой, если нельзя выделить нужный объём памяти.

### Схема перевыделения памяти для очереди на основе динамического массива:

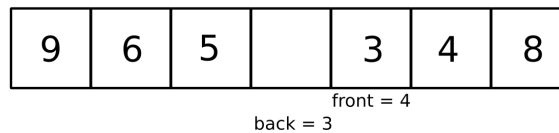
Очередь будет считаться заполненной:

- Если `front == 0`, а `back == capacity - 1`
- Или если `front != 0`, а `front - back == 1`. (А не `front - back == 0`, потому что при полном заполнении вместимости `front` будет равняться `back` и мы не сможем понять полная эта очередь или пустая).

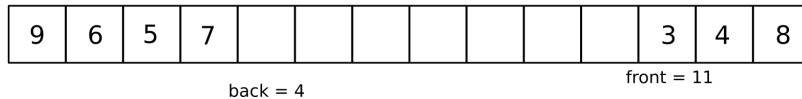
Когда очередь заполнена и мы хотим добавить в неё ещё один элемент, то её нужно увеличить. Делается это так, как представлено на схеме ниже:



*enqueue(&b, 5);*



*enqueue(&b, 7);*



- Если `front == 0`, то нужно просто увеличить очередь с помощью `realloc`.
- Если `front != 0`, то нужно ещё и перекопировать хвост очереди в конец и изменить `front`.