Домашнее задание

Очередь

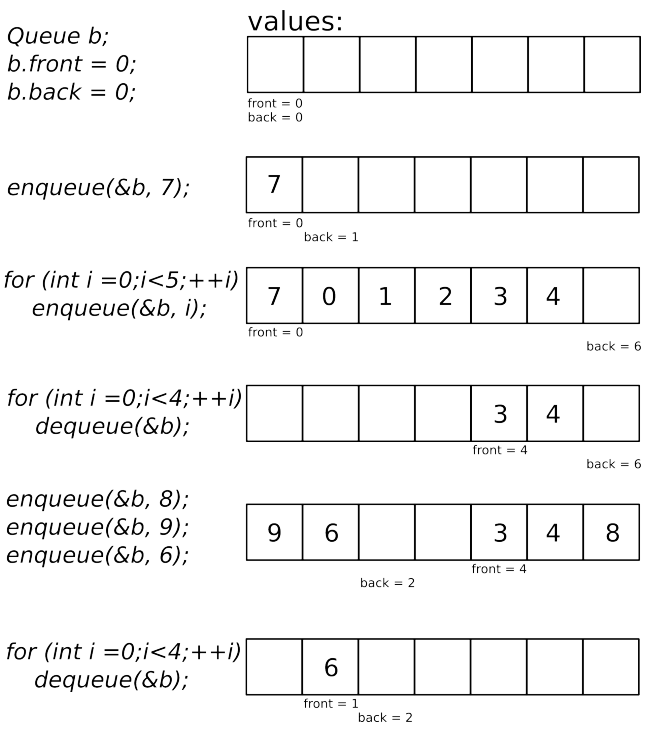
```
#define CAPACITY 7
typedef int Data;

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

// ......

int main()
{
    Queue a;
    a.init();
    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:



Очередь со статическим массивом. Задачи:

1. Написать функцию `void enqueue(Queue* q, Data x)`.
2. Написать функцию `Data dequeue(Queue* q)`.
3. Написать функцию `void queue_init(Queue* q)`. Протестируйте очередь: проверьте, что выведет программа, написанная выше.
4. Написать функцию `int queue_is_empty(Queue* q)`, которая возвращает 1 если очередь пуста и 0 иначе.
5. Написать функцию `int queue_get_size(Queue* q)`, которая возвращает число элементов в очереди (не capacity!).
6. Написать функцию `int queue_is_full(Queue* q)`, которая возвращает 1 если очередь заполнена и 0 иначе.
7. Написать функции `Data queue_get_front(Queue* s)` и `Data queue_get_back(Queue* s)`, которые возвращают элементы, находящиеся в начале и в конце очереди соответственно, но не изменяют очередь.
8. Написать функцию `void queue_print(Queue* s)`, 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. Пример вызова: exit(1);
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()`

Очередь с динамическим массивом. Задачи:

Описание такой очереди выглядит следующим образом:

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

11. Скопируйте код очереди со статическим массивом в новый файл и измените описание структуры как показано выше. Define-макрос CAPACITY больше не нужен, его можно удалить.

12. 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.

13. Измените функцию `void enqueue(Queue* q)`. Теперь, при заполнении очереди должно происходить перевыделение памяти с помощью функции realloc(). 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.

14. Добавьте функцию `void queue_destroy(Queue* q)`, которая будет освобождать память, выделенную под массив values.

15. Протестируйте очередь: в очередь добавляется много элементов ($\gg 10^3 >$ initial_capacity). Программа **не** должна напечатать сообщение об ошибке (если только совокупный размер элементов не превышает размер доступной оперетивной памяти).

16.* (Необязательная задача) 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. Как правильно пользовать realloc() можно посмотреть по следующе ссылке:
https://stackoverflow.com/questions/21006707/proper-usage-of-realloc