Wrap around in C

Ying Pei Lin

Fall 2024

Implementing a queue by array

To implement a queue by array with wrap around, we need to track the position of the first and last elements in the queue. Also, we need to track the number of elements in the queue in order to increase the size of the queue when it is full to achieve dynamic memory allocation.

Below is the implementation of the queue by array with wrap around.

Initialization

Upon initialization, we allocate memory for the queue structure and the array with the default size. The count, first, and last positions are set to zero at the beginning.

```
#define DEFAULT_SIZE 10

queue* create_queue() {
  queue* q = (queue*)malloc(sizeof(queue));
  q->size = DEFAULT_SIZE;
  q->queue = (int*)malloc(q->size * sizeof(int));
  q->count = 0;
  q->first = 0;
  q->last = 0;
  return q;
}
```

Enqueue

Before adding an element to the queue, we check if the queue is full. If the queue is full, we create a new queue with double the size and copy the elements to the new queue, then free the old queue.

Later we put the new element to the last position of the queue and increment the last position by one. If the last position is at the end of the queue, we set it to the beginning of the queue.

```c
void enqueue(queue* q, int data) {

  // Check if the queue is full
  if (q->count == q->size) {
    printf("Overflow, increasing the size of the queue\n");

    // Create a new queue with double the size
    q->size *= 2;
    int* copy = (int*)malloc(q->size * sizeof(int));

    // Copy the elements to the new queue
    int index;
    for (int i = 0; i < q->count; i++) {
      index = (q->first + i) % q->size;
      copy[i] = q->queue[index];
    }

    // Update the old queue with the new queue
    q->first = 0;
    q->last = q->count;
    free(q->queue);
    q->queue = copy;
  }

  // Add the new element to the last position
  q->queue[q->last] = data;
  q->last = (q->last + 1) % q->size;
  q->count++;
}
```

Dequeue

To remove an element from the queue, we first check whether the queue is empty. If the queue is empty, we print a message and return. Otherwise, we get the data from the position of first index, decrease the count of the queue by one, and increment the first index by one. If the first index is at the end of the queue, we set it to the beginning of the queue by using the modulo operator.

```c
void dequeue(queue* q) {
  if (q->count == 0) {
    printf("Empty\n");
    return;
  }
```

```
    // Get the data from the first index
    int data = q->queue[q->first];
    printf("Dequeued: %d\n", data);

    // Update the first index
    q->first = (q->first + 1) % q->size;

    // Decrease the count of the queue
    q->count--;
}
```