```c
/*
 * Modified: Ather, Faiz
 * The University of Queensland
 * Dated: 19th July 2022 21:00:00
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

typedef struct fifo {
    int size;
    int empty;
    int produce;
    int consume;
    char *contents[];
} fifo;

/*
 * Create a new string FIFO object that can contain up to size elements.
 */
struct fifo *fifo_new(int size) {
    struct fifo *container = NULL;

    if (size < 0) { return NULL; }

    container = (struct fifo *)calloc(1, \
      sizeof(struct fifo) + (sizeof(char *) * (unsigned long)size));
    if (container == NULL) {
        fprintf(stderr, "ERR: calloc %d %s\n", errno, strerror(errno));
        return NULL;
    }
    container->size = size;
    container->empty = (size > 0) ? 1 : 0;

    return container;
}

/*
 * Free a FIFO object and all its contents.
 */
void fifo_free(struct fifo *fifo) {
    if (fifo == NULL) { return; }

    int curr = fifo->consume, end = fifo->produce;
    if (curr == end && fifo->empty == 0) {
        curr = 0;
        end = 0;
    }
    while (curr != end || fifo->empty == 0) {
        if (fifo->contents[curr] != NULL) {
            bzero(fifo->contents[curr], strlen(fifo->contents[curr]));
            free(fifo->contents[curr]);
        }
        curr = (curr + 1) % fifo->size;
        if (curr == end) { fifo->empty = 1; }
    }
    bzero(fifo->contents, sizeof(char *) * fifo->size);
    bzero(fifo, sizeof(struct fifo));
    free(fifo);
}
```

```c
/*
 * Push a string into the FIFO.
 * Returns whether there was space in the FIFO to store the string.
 * If successful, the FIFO stores a copy of the string dynamically allocated on the heap.
 * If unsuccessful, the FIFO remains unchanged and no memory is allocated.
 */
int fifo_push(struct fifo *fifo, const char *str) {
    if (fifo == NULL) { return 0; }

    size_t str_sz = 0;
    char *str_cpy = NULL;
    int space = 0;

    space = (((fifo->consume != fifo->produce) || \
      (fifo->size > 0 && (fifo->consume == fifo->produce) && fifo->empty == 1)) ? 1 : 0);
    if (str == NULL || space <= 0) { return space; }

    str_sz = strlen(str) + 1;
    str_cpy = (char *)calloc(str_sz, sizeof(char));
    if (str_cpy == NULL) {
        fprintf(stderr, "ERR: calloc %d %s\n", errno, strerror(errno));
        return space;
    }
    memcpy(str_cpy, str, str_sz);
    fifo->contents[fifo->produce] = str_cpy;
    fifo->produce = (fifo->produce + 1) % fifo->size;
    if (fifo->empty == 1) { fifo->empty = 0; }

    return space;
}

/*
 * Pull a string from the FIFO.
 * Returns NULL if the FIFO is empty.
 * If the returned value is not NULL, the caller takes ownership of the string and
 * is responsible for freeing it.
 */
char *fifo_pull(struct fifo *fifo) {
    if (fifo == NULL) return NULL;

    char *str = NULL;
    int space = 0;

    space = ((((fifo->consume == fifo->produce) && fifo->empty == 1) || \
      fifo->size == 0) ? 0 : 1);
    if (space <= 0) { return NULL; }

    str = fifo->contents[fifo->consume];
    fifo->contents[fifo->consume] = NULL;
    fifo->consume = (fifo->consume + 1) % fifo->size;
    if (fifo->consume == fifo->produce) { fifo->empty = 1; }

    return str;
}

void fifo_dump(struct fifo *fifo) {
    char *str;
    while(str = fifo_pull(fifo), str) {
        printf("%s\n", str);
        free(str);
    }
}
```

```c
#define TEST(condition) if(!(condition)) { printf("TEST FAILED\n"); return 1; }

int main() {
    struct fifo *fifo;
    char *str;

    /* === MY Tests ==== */
    fifo = fifo_new(0);
    TEST(!fifo_push(fifo, "a"));
    str = fifo_pull(fifo);
    TEST(str == NULL);
    TEST(fifo->consume == fifo->produce && fifo->empty == 0 && fifo->produce == 0);
    free(fifo);

    fifo = fifo_new(1);
    TEST(fifo_push(fifo, "a"));
    TEST(!fifo_push(fifo, "a"));
    TEST(fifo->consume == fifo->produce && fifo->empty == 0 && fifo->produce == 0);
    str = fifo_pull(fifo);
    TEST(fifo->consume == fifo->produce && fifo->empty == 1 && fifo->produce == 0);
    TEST(!strcmp(str, "a"));
    free(str);
    fifo_dump(fifo);
    fifo_free(fifo);

    fifo = fifo_new(2);
    TEST(fifo_push(fifo, "a"));
    TEST(fifo_push(fifo, "a"));
    TEST(!fifo_push(fifo, "a"));
    TEST(fifo->consume == fifo->produce && fifo->empty == 0 && fifo->produce == 0);
    str = fifo_pull(fifo);
    TEST(!strcmp(str, "a"));
    free(str);
    str = fifo_pull(fifo);
    TEST(!strcmp(str, "a"));
    free(str);
    TEST(fifo->consume == fifo->produce && fifo->empty == 1 && fifo->produce == 0);
    TEST(fifo_push(fifo, "b"));
    TEST(fifo->consume == 0 && fifo->empty == 0 && fifo->produce == 1);
    str = fifo_pull(fifo);
    TEST(fifo->consume == 1 && fifo->empty == 1 && fifo->produce == 1);
    TEST(!strcmp(str, "b"));
    free(str);
    TEST(fifo_push(fifo, "c"));
    TEST(fifo_push(fifo, "c"));
    TEST(!fifo_push(fifo, "c"));
    TEST(fifo->consume == fifo->produce && fifo->empty == 0 && fifo->produce == 1);
    str = fifo_pull(fifo);
    TEST(!strcmp(str, "c"));
    free(str);
    str = fifo_pull(fifo);
    TEST(fifo->consume == fifo->produce && fifo->empty == 1 && fifo->produce == 1);
    TEST(!strcmp(str, "c"));
    free(str);
    fifo_dump(fifo);
    fifo_free(fifo);

    fifo = fifo_new(3);
    TEST(fifo_push(fifo, "a"));
    TEST(fifo_push(fifo, "a"));
    str = fifo_pull(fifo);
    TEST(!strcmp(str, "a"));
    free(str);
    str = fifo_pull(fifo);
    TEST(!strcmp(str, "a"));
    free(str);
    TEST(fifo->consume == fifo->produce && fifo->empty == 1 && fifo->produce == 2);
    TEST(fifo_push(fifo, "b"));
    TEST(fifo_push(fifo, "b"));
    str = fifo_pull(fifo);
    TEST(!strcmp(str, "b"));
    free(str);
    str = fifo_pull(fifo);
    TEST(!strcmp(str, "b"));
    free(str);
    TEST(fifo->consume == fifo->produce && fifo->empty == 1 && fifo->produce == 1);
    TEST(fifo_push(fifo, "c"));
    TEST(fifo_push(fifo, "c"));
    TEST(fifo_push(fifo, "c"));
    TEST(!fifo_push(fifo, "d"));
    TEST(fifo->consume == fifo->produce && fifo->empty == 0 && fifo->produce == 1);
    fifo_dump(fifo);
    fifo_free(fifo);

    /* === YOUR Tests ==== */
    fifo = fifo_new(4);
    TEST(fifo_push(fifo, "hello"));
    TEST(fifo_push(fifo, "world"));
    fifo_dump(fifo);
    fifo_free(fifo);

    fifo = fifo_new(4);
    TEST(fifo_push(fifo, "elem1"));
    TEST(fifo_push(fifo, "elem2"));
    TEST(fifo_push(fifo, "elem3"));
    TEST(fifo_push(fifo, "elem4"));
    fifo_dump(fifo);
    TEST(fifo_push(fifo, "A"));
    fifo_dump(fifo);
    TEST(fifo_push(fifo, "X"));
    TEST(fifo_push(fifo, "Y"));
    TEST(fifo_push(fifo, "Z"));
    TEST(fifo_push(fifo, "T"));
    TEST(!fifo_push(fifo, "U"));
    fifo_dump(fifo);
    fifo_free(fifo);

    fifo = fifo_new(4);
    TEST(fifo_push(fifo, "elem1"));
    TEST(fifo_push(fifo, "elem2"));
    TEST(fifo_push(fifo, "elem3"));
    TEST(fifo_push(fifo, "elem4"));
    fifo_free(fifo);

    fifo = fifo_new(4);
    TEST(fifo_push(fifo, "elem1"));
    TEST(fifo_push(fifo, "elem2"));
    str = fifo_pull(fifo);
    TEST(!strcmp(str, "elem1"));
    free(str);
    TEST(fifo_push(fifo, "elem3"));
    TEST(fifo_push(fifo, "elem4"));
    str = fifo_pull(fifo);
    TEST(!strcmp(str, "elem2"));
    free(str);
    str = fifo_pull(fifo);
```

```c
    TEST(!strcmp(str, "elem3"));
    free(str);
    str = fifo_pull(fifo);
    TEST(!strcmp(str, "elem4"));
    free(str);
    fifo_free(fifo);

    return 0;
}
```