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

struct fifo {
        int size;
        int empty;
        int produce;//int produce;
        int consume; // int consume;
        char** items;//contents;
};

void fifo_free(struct fifo* fifo);
int isEmpty(struct fifo* fifo);

// Create a new string FIFO object that can contain up to size
elements.
struct fifo* fifo_new(int size) {
        // COMPLETE THIS PART
        struct fifo* my_fifo = (struct fifo*)malloc(sizeof(struct
fifo*));
        if (my_fifo != NULL) {
                my_fifo->consume = 0;
                my_fifo->produce = 0;
                my_fifo->size = size + 1;
                my_fifo->items = (char**)malloc(sizeof(char*) *
(my_fifo->size));
                my_fifo->empty = size;
        }
        return my_fifo;
}


// Free a FIFO object and all its contents.
void fifo_free(struct fifo* fifo) {

        while (isEmpty(fifo) == 0) {
                free(fifo->items[fifo->consume]);
                if (fifo->consume == fifo->produce) {
                        fifo->consume = 0;
                        fifo->produce = 0;
                }
                // Q has only one element, so we reset the
                // queue after dequeing it. ?
                else {
                        fifo->consume = (fifo->consume + 1) %
fifo->size;
                }
                //printf("\n Deleted element -> %d \n", element);
        }
        free(fifo->items);
        free(fifo);
}
```

```
// 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.


// Check if the queue is full
int isFull(struct fifo* fifo) {
        if (fifo->consume == (fifo->produce + 1) % fifo->size)
                return 1;
        return 0;
}

// Check if the queue is empty
int isEmpty(struct fifo* fifo) {
        if (fifo->consume == fifo->produce)
                return 1;
        return 0;
}


int fifo_push(struct fifo* fifo, const char* str) {
        if (isFull(fifo) == 0) {
                int new_rear = (fifo->produce + 1) % fifo->size;
                fifo->items[fifo->produce] = malloc(strlen(str) +
1);
                strcpy(fifo->items[fifo->produce], str);
                //printf("\n Inserted -> %d", fifo->element);
                fifo->produce = new_rear;
                return 1;
        }
        else {
                return 0;
        }
}

// 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 (isEmpty(fifo) == 0) {
                int len = strlen(fifo->items[fifo->consume]);
                char* str = (char*)malloc(len + 1);
                strcpy(str, fifo->items[fifo->consume]);


                free(fifo->items[fifo->consume]);
                fifo->consume = (fifo->consume + 1) % fifo->size;
                return str;
```

```
        }
        else {
                return 0;
        }
}

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

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

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

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

        return 0;
}
```