#include <stdio.h>#include <stdlib.h>#include <stdbool.h>

struct Node { // A node in a circular list

char value; // The value

struct Node \*prev; // The previous node

struct Node \*next; // The next node

};

struct CircularList { // A circular list

struct Node \*cursor; // The current node

unsigned int num\_elements; // The number of elements

};

struct CircularList circular\_create() {

struct CircularList cl;

cl.cursor = NULL;

cl.num\_elements = 0;

return cl;

}

bool circular\_is\_empty(const struct CircularList \*cl) {

return cl->num\_elements == 0;

}

void circular\_print(const struct CircularList \*cl, unsigned int n, bool forward) {

printf("( ");

if (circular\_is\_empty(cl)) {

printf(" )");

return;

}

struct Node \*current = cl->cursor;

for (unsigned int i = 0; i < n; ++i) {

printf("%c ", current->value);

current = forward ? current->next : current->prev;

}

printf(")");

}

value The value to be inserted

\*/

void circular\_insert(struct CircularList \*cl, char value) {

struct Node \*node = malloc(sizeof(struct Node));

node->value = value;

if (circular\_is\_empty(cl)) {

node->next = node;

node->prev = node;

} else {

node->next = cl->cursor;

node->prev = cl->cursor->prev;

cl->cursor->prev->next = node;

cl->cursor->prev = node;

}

cl->cursor = node;

++cl->num\_elements;

}

char circular\_pop(struct CircularList \*cl) {

if (!circular\_is\_empty(cl)) {

struct Node \*node = cl->cursor;

char value = cl->cursor->value;

if (cl->num\_elements == 1) {

cl->cursor = NULL;

} else {

cl->cursor->prev->next = cl->cursor->next;

cl->cursor->next->prev = cl->cursor->prev;

cl->cursor = cl->cursor->next;

}

free(node);

--cl->num\_elements;

return value;

} else {

fprintf(stderr, "Error: cannot pop from empty circular list\n");

return '?';

}

}

void circular\_shift(struct CircularList \*cl, int i) {

if (!circular\_is\_empty(cl)) {

while (i != 0) {

if (i > 0) {

cl->cursor = cl->cursor->next;

--i;

} else {

cl->cursor = cl->cursor->prev;

++i;

}

}

}

}

void circular\_delete(struct CircularList \*cl) {

while (!circular\_is\_empty(cl)) {

circular\_pop(cl);

}

}

int main() {

struct CircularList cl = circular\_create();

printf("Initially, the circular list is empty: ");

circular\_print(&cl, cl.num\_elements, true);

printf("\nInserting the characters A, C, A, T, T, A, G: ");

char \*gattaca = "ACATTAG";

for (unsigned int i = 0; i < 7; ++i) {

circular\_insert(&cl, gattaca[i]);

}

circular\_print(&cl, cl.num\_elements, true);

printf("\nPrinted backward: ");

circular\_print(&cl, cl.num\_elements, false);

printf("\nPrinting 18 elements: ");

circular\_print(&cl, 18, true);

printf("\nPrinting 18 elements backward: ");

circular\_print(&cl, 18, false);

printf("\nShifted by 2: ");

circular\_shift(&cl, 2);

circular\_print(&cl, cl.num\_elements, true);

printf("\nShifted by -3: ");

circular\_shift(&cl, -3);

circular\_print(&cl, cl.num\_elements, true);

printf("\nRemoving current node: ");

circular\_pop(&cl);

circular\_print(&cl, cl.num\_elements, true);

printf(" forward and ");

circular\_print(&cl, cl.num\_elements, false);

printf(" backward\n");

printf("Removing 4 nodes: ");

for (unsigned int i = 0; i < 4; ++i) {

circular\_pop(&cl);

}

circular\_print(&cl, cl.num\_elements, true);

printf("\nInserting C: ");

circular\_insert(&cl, 'C');

circular\_print(&cl, cl.num\_elements, true);

printf("\n");

circular\_delete(&cl);

return 0;

}

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

#define BUFFER\_SIZE 256

void lire\_fichier(const char \*nom\_fichier) {

FILE \*file = fopen(nom\_fichier, "r");

if (file == NULL) {

perror("Erreur lors de l'ouverture du fichier");

return;

}

char buffer[BUFFER\_SIZE];

printf("Contenu du fichier:\n");

while (fgets(buffer, BUFFER\_SIZE, file) != NULL) {

printf("%s", buffer);

}

fclose(file);

}

void ecrire\_fichier(const char \*nom\_fichier, const char \*contenu) {

FILE \*file = fopen(nom\_fichier, "w");

if (file == NULL) {

perror("Erreur lors de l'ouverture du fichier");

return;

}

fprintf(file, "%s\n", contenu);

fclose(file);

}

int main() {

const char \*nom\_fichier = "example.txt";

// Lire et afficher le contenu du fichier

lire\_fichier(nom\_fichier);

// Modifier le contenu du fichier

const char \*nouveau\_contenu = "Hello, World!";

ecrire\_fichier(nom\_fichier, nouveau\_contenu);

printf("\nContenu du fichier après modification:\n");

lire\_fichier(nom\_fichier);

return 0;

}

MAKEFILE

# Nom de l'exécutable

TARGET = programme

# Liste des fichiers source

SRCS = main.c

# Compilateur et options de compilation

CC = gcc

CFLAGS = -Wall -Wextra -std=c11

# Règle par défaut

all: $(TARGET)

# Règle pour créer l'exécutable

$(TARGET): $(SRCS)

$(CC) $(CFLAGS) -o $(TARGET) $(SRCS)

# Règle pour nettoyer les fichiers générés

clean:

rm -f $(TARGET) \*.o

# Phony targets pour éviter des conflits avec des fichiers ayant le même nom

.PHONY: all clean

BATS:

# Test de la modification du fichier

@test "File content change" {

# Compiler le programme

make

# Exécuter le programme

run ./programme

# Vérifier que le contenu a été changé

[ "$status" -eq 0 ]

[[ "${lines[3]}" == "Hello, World!" ]]

}

# Nettoyer après les tests

teardown() {

rm -f example.txt programme

}