**Inc- header.h**

#ifndef \_\_HEADER\_\_

#define \_\_HEADER\_\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Macro definitions

#define SUCCESS 1

#define FAILURE 0

#define MAX\_STRING\_LENGTH 80 // Max string length

// Node structure for the queue

typedef struct node\_q {

int data; // Integer data

char \*str\_data; // String data read from user

struct node\_q \*next; // Pointer to next node

} node;

// Function declarations

void enqueue(node \*\*front, node \*\*rear, int value, const char \*str);

int dequeue(node \*\*front, node \*\*rear, int \*value, char \*\*str);

int isempty(node \*front);

void display(node \*front);

int free\_queue(node \*front, node \*rear);

#endif

**Makefile:**

SRC = ./../src/

OBJ = ./../obj/

BIN = ./../bin/

GFLAGS = gcc

INC = ./../include/

CFLAGS = -c -g -Wall

OFLAGS = -lm -o

IFLAGS = -I./../include/

VFLAGS = v --tool=memcheck --leak-check=yes --show-reachable=yes

EXECS = $(BIN)final

$(BIN)final :$(OBJ)main.o $(OBJ)function.o

$(GFLAGS) $(OFLAGS)final $(OBJ)main.o $(OBJ)function.o

mv final $(BIN)

$(OBJ)main.o:$(SRC)main.c $(INC)header.h

$(GFLAGS) $(CFLAGS) $(SRC)main.c $(IFLAGS)

mv main.o $(OBJ)

$(OBJ)function.o:$(SRC)function.c $(INC)header.h

$(GFLAGS) $(CFLAGS) $(SRC)function.c $(IFLAGS)

mv function.o $(OBJ)

clean:

rm $(OBJ)\*

rm $(BIN)final

valgrind:$(EXECS)

valgrind $(VFLAGS) $(BIN)final

**src – function.c**

#include "header.h"

// Function to check if the queue is empty

int isempty(node \*front) {

return (front == NULL) ? SUCCESS : FAILURE;

}

// Function to insert a node in the queue

void enqueue(node \*\*front, node \*\*rear, int value, const char \*str) {

// Allocate memory for the new node

node \*newnode = (node \*)malloc(sizeof(node));

if (newnode == NULL) {

printf("Memory not available\n");

exit(FAILURE);

}

// Assign integer data

newnode->data = value;

// Allocate memory for the string and copy the user input

newnode->str\_data = (char \*)malloc((strlen(str) + 1) \* sizeof(char));

if (newnode->str\_data == NULL) {

printf("Memory not available for string\n");

free(newnode); // Free the node memory if string allocation fails

exit(FAILURE);

}

strcpy(newnode->str\_data, str);

// Set the next pointer to NULL

newnode->next = NULL;

// Insert at the rear of the queue

if (isempty(\*front)) {

\*front = newnode;

\*rear = newnode;

} else {

(\*rear)->next = newnode;

\*rear = newnode;

}

}

// Function to delete a node from the front of the queue

int dequeue(node \*\*front, node \*\*rear, int \*value, char \*\*str) {

if (isempty(\*front)) {

printf("Queue Underflow\n");

return FAILURE;

}

node \*temp\_node = \*front;

\*value = temp\_node->data; // Get integer value

\*str = temp\_node->str\_data; // Get string data

// If only one node is present in the queue

if (\*front == \*rear) {

\*front = NULL;

\*rear = NULL;

} else {

\*front = (\*front)->next;

}

free(temp\_node->str\_data); // Free string data

free(temp\_node); // Free the node itself

return SUCCESS;

}

// Function to display the queue contents

void display(node \*front) {

if (isempty(front)) {

printf("The Queue is empty\n");

return;

}

node \*current = front;

printf("Queue contents:\n");

while (current != NULL) {

printf("Integer: %d, String: %s\n", current->data, current->str\_data);

current = current->next;

}

}

// Function to free the memory of the entire queue

int free\_queue(node \*front, node \*rear) {

if (isempty(front)) {

return FAILURE;

}

node \*current = front;

while (current != NULL) {

node \*next\_node = current->next;

free(current->str\_data); // Free the string data first

free(current); // Free the node memory

current = next\_node;

}

rear = NULL;

return SUCCESS;

}

**Src- main.c**

#include "header.h"

// Main function to drive the queue operations

int main(void) {

node \*front = NULL, \*rear = NULL;

int status, value;

char input\_string[MAX\_STRING\_LENGTH]; // Buffer to hold user input string

char \*str = NULL;

while (1) {

printf("\n1.Insert\n2.Remove\n3.Display\n4.Exit\n");

int choice;

scanf("%d", &choice);

getchar(); // To consume the newline character after scanf

switch (choice) {

case 1: // Insert a new element

printf("Enter an integer value: ");

scanf("%d", &value);

getchar(); // To consume the newline character after integer input

printf("Enter a string (max %d characters): ", MAX\_STRING\_LENGTH);

fgets(input\_string, sizeof(input\_string), stdin);

input\_string[strcspn(input\_string, "\n")] = '\0'; // Remove newline character

enqueue(&front, &rear, value, input\_string);

break;

case 2: // Remove an element

status = dequeue(&front, &rear, &value, &str);

if (status == SUCCESS) {

printf("Element removed: Integer = %d, String = %s\n", value, str);

free(str); // Free the string memory after use

}

break;

case 3: // Display the queue

display(front);

break;

case 4: // Exit

status = free\_queue(front, rear);

if (status == SUCCESS) {

printf("Memory freed.\n");

} else {

printf("Queue was already empty.\n");

}

exit(0);

break;

default:

printf("Invalid option. Please try again.\n");

break;

}

}

return 0;

}