**Header.h**

#ifndef \_\_HEADER\_\_

#define \_\_HEADER\_\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MACRO DECLARATIONS \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SUCCESS 1

#define FAILURE 0

#define END -1

#define MAX\_QUEUE\_SIZE 5 // Can be dynamically set at runtime

/\* Data Type for Queue ADT \*/

typedef struct queue

{

char \*\*data; // Dynamic array to store strings

int front;

int rear;

} queue\_t;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function declarations \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void init\_queue(queue\_t \*q, int capacity);

int check\_empty(queue\_t \*q);

int check\_full(queue\_t \*q);

int enqueue(queue\_t \*q, char \*elem);

int dequeue(queue\_t \*q, char \*\*p\_elem);

void display(queue\_t \*q);

#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 NAME: init\_queue

\* DESCRIPTION: Initialize queue to empty

\* RETURNS: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void init\_queue(queue\_t \*q, int capacity)

{

q->data = (char \*\*)malloc(capacity \* sizeof(char \*)); // Allocate memory for strings

for (int i = 0; i < capacity; i++) {

q->data[i] = (char \*)malloc(100 \* sizeof(char)); // Allocate memory for each string

}

q->front = END;

q->rear = END;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION NAME: check\_empty

\* DESCRIPTION: Check if the queue is empty

\* RETURNS: SUCCESS or FAILURE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int check\_empty(queue\_t \*q)

{

if (END == q->front)

{

return SUCCESS;

}

else

{

return FAILURE;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION NAME: check\_full

\* DESCRIPTION: Check if the queue is full

\* RETURNS: SUCCESS or FAILURE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int check\_full(queue\_t \*q)

{

if ((q->rear + 1) % MAX\_QUEUE\_SIZE == q->front)

{

return SUCCESS;

}

else

{

return FAILURE;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION NAME: enqueue

\* DESCRIPTION: Inserts an element (string) into the queue

\* RETURNS: SUCCESS or FAILURE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int enqueue(queue\_t \*q, char \*elem)

{

if (check\_full(q))

{

return FAILURE; // Queue full

}

if (check\_empty(q))

{

q->front = 0;

q->rear = 0;

}

else

{

q->rear = (q->rear + 1) % MAX\_QUEUE\_SIZE;

}

strcpy(q->data[q->rear], elem); // Insert the string at the rear

return SUCCESS;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION NAME: dequeue

\* DESCRIPTION: Removes an element (string) from the queue

\* ARGUMENTD: pointer to the returned element (string)

\* RETURNS: SUCCESS or FAILURE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int dequeue(queue\_t \*q, char \*\*p\_elem)

{

if (check\_empty(q))

{

return FAILURE; // Queue empty

}

\*p\_elem = q->data[q->front]; // Return the string at the front

if (q->front == q->rear)

{

q->front = END;

q->rear = END;

}

else

{

q->front = (q->front + 1) % MAX\_QUEUE\_SIZE;

}

return SUCCESS; // Success

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION NAME: display

\* DESCRIPTION: Displays the contents of the queue

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display(queue\_t \*q)

{

if (check\_empty(q))

{

printf("Queue is empty\n");

return;

}

int index = q->front;

while (index != q->rear)

{

printf("Queue[%d]: %s\n", index, q->data[index]);

index = (index + 1) % MAX\_QUEUE\_SIZE;

}

printf("Queue[%d]: %s\n", q->rear, q->data[q->rear]);

}

**Src- Main.c**

#include "header.h"

int main()

{

queue\_t q1;

int option;

int status; // success/failure return

char elem[100]; // buffer to store user input string

int queue\_size;

printf("Enter the maximum size of the queue: ");

scanf("%d", &queue\_size);

// Initialize the Queue with user-defined size

init\_queue(&q1, queue\_size);

// Main Input loop (or driver loop)

while(1)

{

printf("Please enter your choice.\n");

printf("1) Insert an element\n");

printf("2) Remove an element\n");

printf("3) Check Empty\n");

printf("4) Check Full\n");

printf("5) Debug Print the queue data\n");

printf("6) Exit\n");

scanf("%d", &option);

switch (option)

{

case 1: /\* Insert an element \*/

printf("Enter the string to insert: ");

getchar(); // to capture the newline character left by previous input

fgets(elem, sizeof(elem), stdin);

elem[strcspn(elem, "\n")] = 0; // Remove trailing newline

status = enqueue(&q1, elem);

if (SUCCESS == status)

{

printf(" Element inserted Successfully \n");

}

else

{

printf(" Queue full, cannot insert \n");

}

break;

case 2: /\* Remove the front element \*/

status = dequeue(&q1, &elem);

if (SUCCESS == status)

{

printf(" Element '%s' removed Successfully \n", elem);

}

else

{

printf(" Queue Empty, cannot remove \n");

}

break;

case 3: /\* Check Empty \*/

status = check\_empty(&q1);

if (SUCCESS == status)

{

printf(" Queue is Empty \n");

}

else

{

printf(" Queue is not Empty \n");

}

break;

case 4: /\* Check Full \*/

status = check\_full(&q1);

if (SUCCESS == status)

{

printf(" Queue is Full \n");

}

else

{

printf(" Queue is not Full \n");

}

break;

case 5: /\* Debug Print \*/

display(&q1);

break;

case 6: /\* Exit\*/

exit(0);

break;

}

}

exit(0);

}