```c
/**
* @file common_variable.h
* @brief contains the variables used across other files
*
*
* @author Steve and Swarupa
* @date Nov 7, 2018
*
*/
//*****************************************************************************
// Include files
//*****************************************************************************
#ifndef _COMMON_VARIABLE_H_
#define  _COMMON_VARIABLE_H_
#include <stdint.h>

//*****************************************************************************
// Macros
//*****************************************************************************

//---------------------------------------------------------------------------
// Mode selection in FRDM : POLLING OR INTERRUPT

//define FRDM
//#define INTERRUPT
#define POLLING

//#define DEBUG
//---------------------------------------------------------------------------


// Print function

#ifdef FRDM
	#define PRINT send_to_console_str
#endif

#ifdef LINUX
	#define PRINT printf
#endif

//*****************************************************************************
// Globals
//*****************************************************************************

//Error handling enums
typedef enum status_t
{
	NULL_PTR = -5,
	OVERFLOW = -4,
	EMPTY = -3,
	BUFFER_NOT_INITIALISED = -2,
	ERROR = -1,
	SUCCESS = 1,
}status;
```

```c
// Node for Circular Linked List
struct node
{
	uint8_t data;
	struct node * link;
};


//Circular buffer declaration
typedef struct
{
	struct node* front_CB;
	struct node* rear_CB;
	struct node *head;
	struct node *tail;
	int32_t length_CB;
	int32_t max_size;
	int8_t flag_init;
}CB;



CB RX_buffer ;

#endif

/**
* @file clear_buffer.h
* @brief An abstraction for clear_buffer.c
*
* This header file provides declarations of clear_buffer.c
*
* @author Steve and Swarupa
* @date Nov 7, 2018
*
*/
//*****************************************************************************
// Include files
//*****************************************************************************

#ifndef _CLEAR_BUFFER_H_
#define _CLEAR_BUFFER_H_


#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "common_variable.h"

#ifdef FRDM
#include "uart.h"

#endif
```

```
//*****************************************************************************
// function prototypes
//*****************************************************************************
status clear_buffer(CB *);
int8_t IsEMPTY(CB *);

#endif


/**
* @file delete_data.h
* @brief An abstraction for delete_data.c
*
* This header file provides declarations of delete_data.c
*
* @author Steve and Swarupa
* @date Nov 6, 2018
*
*/
//*****************************************************************************
// Include files
//*****************************************************************************

#ifndef _DELETE_CB_H_
#define _DELETE_CB_H_


#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "common_variable.h"

#ifdef FRDM
#include "uart.h"
#endif

//*****************************************************************************
// function prototypes
//*****************************************************************************
status delete_CB(CB *);
int8_t IsEMPTY(CB *);

#endif

/**
* @file init_CB.h
* @brief An abstraction for init_CB.c
*
* This header file provides declarations of init_CB.c
*
* @author Steve and Swarupa
* @date Nov 6, 2018
*
*/
//*****************************************************************************
```

```
// Include files
//*****************************************************************************

#ifndef _CIRCULAR_BUFFER_H_
#define  _CIRCULAR_BUFFER_H_
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "common_variable.h"

#ifdef FRDM
#include "uart.h"
#endif

//*****************************************************************************
// function prototypes
//*****************************************************************************


status init_CB(CB *, int32_t);
int8_t IsFULL(CB *);
int8_t IsEMPTY(CB *);
status insert_link(CB *);

#endif

/**
* @file insert_data.h
* @brief An abstraction for insert_data.c
*
* This header file provides declarations of insert_data.c
*
* @author Steve and Swarupa
* @date Nov 6, 2018
*
*/

//*****************************************************************************
// Include files
//*****************************************************************************

#ifndef _INSERT_DATA_H_
#define _INSERT_DATA_H_

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "common_variable.h"

#ifdef FRDM
#include "uart.h"
#endif

//*****************************************************************************
// function prototypes
```

```
//*****************************************************************************
status insert_data(CB *, uint8_t);
int8_t IsEMPTY(CB *);
int8_t IsFULL(CB *);


#endif

/**
* @file main.h
* @brief An abstraction for main.c
*
* This header file provides declarations of main.c
*
* @author Steve and Swarupa
* @date Nov 6, 2018
*
*/

//*****************************************************************************
// Include files
//*****************************************************************************

#ifndef _MAIN_H_
#define _MAIN_H_

#include <stdint.h>
#include <stdio.h>
#include "common_variable.h"

#ifdef FRDM
#include "uart.h"

#include "MKL25Z4.h"

//*****************************************************************************
// Macros for FRDM
//*****************************************************************************

#define MAX_PRIME_NUMBER (9999999)
#define MAX_ASCII (256)
#define ROWS_PATTERN_MAX (5)

//Macros for systick timer
#define SYSTICK_CTRL (*((volatile unsigned long *) (0xE000E010)))
#define SYSTICK_LOAD (*((volatile unsigned long *) (0xE000E014)))
#define SYSTICK_VAL (*((volatile unsigned long *) (0xE000E018)))

#endif

//*****************************************************************************
// Common Macros
//*****************************************************************************
#define SIZE_OF_RX_CB (30)
```

```
//*****************************************************************************
// Function prototypes
//*****************************************************************************

status resize_CB(CB *, int32_t );
status insert_link(CB *);
status insert_data(CB *,uint8_t);
int8_t IsEMPTY(CB *);
int8_t IsFULL(CB *);
status delete_CB(CB *);
status init_CB(CB *,int32_t);
status report_data(CB *);
status clear_buffer(CB *);
status pop_data(CB *,uint8_t *);
void LED_init_IRQ();
void receiver_polling();
void transmitter_polling();

#endif

/**
* @file pop_data.h
* @brief An abstraction for pop_data.c
*
* This header file provides declarations of pop_data.c
*
* @author Steve and Swarupa
* @date Nov 18, 2018
*
*/
//*****************************************************************************
// Include files
//*****************************************************************************

#ifndef _POP_DATA_H_
#define _POP_DATA_H_


#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "common_variable.h"

#ifdef FRDM
#include "uart.h"
#endif



//*****************************************************************************
// function prototypes
//*****************************************************************************
status pop_data(CB *,uint8_t *);
int8_t IsEMPTY(CB *);
```

```
#endif

/**
* @file insert_link.h
* @brief An abstraction for insert_link.c
*
* This header file provides declarations of insert_link.c
*
* @author Steve and Swarupa
* @date Nov 6, 2018
*
*/

//*****************************************************************************
// Include files
//*****************************************************************************

#ifndef _REPORT_DATA_H_
#define _REPORT_DATA_H_

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "common_variable.h"

#ifdef FRDM
#include "uart.h"
#endif
//*****************************************************************************
// function prototypes
//*****************************************************************************
status report_data(CB *);
int8_t IsEMPTY(CB *);

#endif

/**
* @file resize_CB.h
* @brief An abstraction for resize_CB.c
*
* This header file provides declarations of resize_CB.c
*
* @author Steve and Swarupa
* @date Nov 6, 2018
*
*/
//*****************************************************************************
// Include files
//*****************************************************************************

#ifndef _RESIZE_CB_H_
#define _RESIZE_CB_H_
```

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "common_variable.h"

#ifdef FRDM
#include "uart.h"
#endif

//*****************************************************************************
// function prototypes
//*****************************************************************************
status resize_CB(CB *,int32_t );
status insert_link(CB *);

#endif

/*
 * uart.h
 *
 *  Created on: Nov 7, 2018
 *      Author: Swarupa De
 */

//*****************************************************************************
// Include files
//*****************************************************************************

#ifndef INCLUDE_UART_H_
#define INCLUDE_UART_H_

#include "common_variable.h"

//*****************************************************************************
// Macros
//*****************************************************************************

#define UART0_BAUD_RATE (57600)


//*****************************************************************************
// Function Prototypes
//*****************************************************************************
void uartinit();
void RX_interrupt_init();
void send_to_console_str(char []);
void send_to_console(uint8_t);
void receiver_polling();
void transmitter_polling();

status resize_CB(CB *, int32_t );
status insert_link(CB *);
status insert_data(CB *,uint8_t);
int8_t IsEMPTY(CB *);
int8_t IsFULL(CB *);
```

```
status delete_CB(CB *);
status init_CB(CB *,int32_t);
status report_data(CB *);
status clear_buffer(CB *);
status pop_data(CB *,uint8_t *);
void sys_reload();


#endif /* INCLUDE_UART_H_ */

/**
* @file uart.c
*
* This file contains function configuring, tranmitting and receiving part of UART0
*
* @author Steve and Swarupa
* @date Nov 24, 2018
*
*/

//*****************************************************************************
// Include files
//*****************************************************************************
#include "MKL25Z4.h"
#include "uart.h"
#include "main.h"
#include <stdlib.h>
#include <string.h>

//declare rx_buffer and tx_buffer for data manipulation
extern uint8_t data_pop;
extern uint32_t database[256] ;
//variable to store the received data
extern uint8_t data_poll;

//Flag to indicate that a data is received
extern int8_t FLAG_RECV;

//*****************************************************************************
// Function definition
//*****************************************************************************

//*****************************************************************************
// Name        : uartinit
//
// Description : Function to initiate UART0
//
// Arguments   : None
//
// return      : None
//
//*****************************************************************************/
void uartinit()
{
```

```
                 uint16_t baudmoddivisor;

                //Set gate clock  for PORTA
                   SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK;

                   //Enable clock for UART0
                   SIM_SCGC4 |= SIM_SCGC4_UART0_MASK;

                   // Set the alternate function for PORTA as UART0
                   PORTA_PCR1 |= PORT_PCR_MUX(0x2);
                   PORTA_PCR2 |= PORT_PCR_MUX(0x2);

                   //set clock source as MCGPLLCLK/2
                   SIM_SOPT2 &= ~(SIM_SOPT2_PLLFLLSEL_MASK);

                   SIM_SOPT2 |= SIM_SOPT2_UART0SRC(1);
                   //SIM_SOPT2 |= SIM_SOPT2_PLLFLLSEL(1);

                   // Disable tx and rx before editing registers of UART0
                   UART0_C2 |= (UART0_C2_TE(0) | UART0_C2_RE(0));


                   //calculate the baud rate modulo divisor and set the baud rate
                   UART0_BDH &= ~UART0_BDH_SBR_MASK;
                   UART0_BDL &= ~UART0_BDL_SBR_MASK;

                   baudmoddivisor= (uint16_t)((SystemCoreClock)/(UART0_BAUD_RATE
*16));
                   UART0_BDH |= UARTLP_BDH_SBR((baudmoddivisor >> 8)) ;
                   UART0_BDL |= UARTLP_BDL_SBR(baudmoddivisor);

                   //Selecting 8 bit data, No parity
                   UART0_C1 |= UART0_C1_M(0) | UART_C1_PE(0);

                   //selecting one stop bit
                   UART0_BDH |= UART_BDH_SBNS(0);


                   //Enabling the Transmitter enable and receiver enable bit
                   UART0_C2 |= (UART0_C2_TE(1) | UART0_C2_RE(1));



}

//*****************************************************************************
// Function definition
//*****************************************************************************

//*****************************************************************************
// Name        : RX_interrupt_init
//
// Description : Function to activate receive interrupt in UART0
//
// Arguments   : None
```

```
//
// return      : None
//
//*****************************************************************************/

void RX_interrupt_init()
{

	NVIC_EnableIRQ(UART0_IRQn);

	//enabling RIE
	UART0_C2 |= (UART_C2_RIE_MASK);

}

//*****************************************************************************
// Function definition
//*****************************************************************************

//*****************************************************************************
// Name        : send_to_console_str
//
// Description : Function to transmit a string
//
// Arguments   : string to be transmitted
//
// return      : None
//
//*****************************************************************************/
void send_to_console_str(char data[])
{
	for(int i =0; data[i] != '\0'; i++)
	{
		//polling for transmitting
		while(!(UART0_S1 & UART_S1_TDRE_MASK));
		UART0_D = (data[i]);
		while(!(UART0_S1 & UART_S1_TC_MASK));
	}
}

//*****************************************************************************
// Function definition
//*****************************************************************************

//*****************************************************************************
// Name        : send_to_console
//
// Description : Function to transmit a byte
//
// Arguments   : byte to be transmitted
//
// return      : None
//
//*****************************************************************************/
void send_to_console(uint8_t data)
```

```
{
	while(!(UART0_S1 & UART_S1_TDRE_MASK));
	UART0_D = (data);
	while(!(UART0_S1 & UART_S1_TC_MASK));
}



//*****************************************************************************
// RX ISR definition
//*****************************************************************************

//*****************************************************************************
// Name        : UART0_IRQHandler
//
// Description : IRQ for receive interrupt
//
// Arguments   : None
//
// return      : None
//
//*****************************************************************************/
void UART0_IRQHandler()
{
	if((UART0_S1) & (UART0_S1_RDRF_MASK))
		{
			PTB->PCOR = (1<<18); //on red

			//insert byte to circular buffer
			insert_data(&RX_buffer, UART0_D);

			//pop data from circular buffer
			pop_data(&RX_buffer,&data_pop);

			//updating the count database
			database[data_pop] = database[data_pop] + 1;

			//initiating the systick timer
			sys_reload();
			PTB->PSOR = (1<<18); // off red
		}
}

//*****************************************************************************
// Name        : receiver_polling
//
// Description : receiver function when polling mode is chosen
//
// Arguments   : None
//
// return      : None
//
//*****************************************************************************/
void receiver_polling()
{
```

```
	data_poll = UART0_D;
	FLAG_RECV = 1;
}
//*****************************************************************************
// Name        : transmitter_polling
//
// Description : transmitter function when polling mode is chosen
//
// Arguments   : None
//
// return      : None
//
//*****************************************************************************/
void transmitter_polling()
{
	UART0_D = data_poll;
}


//*****************************************************************************
// Name        : resize_CB
//
// Description : Function to resize the existing circular Buffer
//
// Author      : Steve and Swarupa
//
// Date        : Nov 06, 2018
//
// Arguments   : elements_to_add - No. of elements to the added to the circular
buffer inorder to resize it
//
// return      : status
//                          SUCCESS on completion, otherwise ERROR
//
//*****************************************************************************/


//*****************************************************************************
// Include files
//*****************************************************************************
#include "resize_CB.h"

//*****************************************************************************
// Function definition
//*****************************************************************************
status resize_CB(CB *buffer, int32_t elements_to_add)
{
	//checks if it is a null pointer
	if(buffer==NULL)
	{
		PRINT("Null pointer input\r\n");
		return NULL_PTR;
	}

	//check if the number of elements to be added is valid number
```

```
        if(elements_to_add<=0)
        {
                PRINT("elements_to_add input is not valid\r\n");
                return ERROR;
        }

        //check if the buffer is initialised
        else if((buffer->flag_init) != 1)
        {
                PRINT("Buffer not initialized\r\n");
                return BUFFER_NOT_INITIALISED;
        }

        //resizing
        else
        {
                for(int i=1; i<=elements_to_add;i++)
                {
                        if((insert_link(buffer)) == ERROR)
                        {
                                PRINT("No space in memoryr\n");
                                return ERROR;
                        }
          }
     }
        return SUCCESS;
}

//*************************************************************************
// Name        : report_data
//
// Description : Function to report the data present in circular buffer
//
// Author      : Steve and Swarupa
//
// Date        : Nov 06, 2018
//
// Arguments   : No
//
// return      : status
//                          SUCCESS on completion, otherwise ERROR
//
//*************************************************************************/


//*************************************************************************************
// Include files
//*************************************************************************************
#include "report_data.h"

//*************************************************************************************
// Function definition
//*************************************************************************************

status report_data(CB *buffer)
```

```
{
	//checks if it is a null pointer
	if(buffer==NULL)
	{
		PRINT("Null pointer input\r\n");
		return NULL_PTR;
	}

	//check if the buffer is initialised
	if((buffer->flag_init) != 1)
	{
		PRINT("Buffer not initialized\r\n");
		return BUFFER_NOT_INITIALISED;
	}

	//check if the buffer is empty
	else if(IsEMPTY(buffer))
	{
		PRINT("No data to display: Buffer is empty\r\n");
		return EMPTY;
	}


	// report data function
	else
	{
		struct node* temp;
		uint32_t count = 0;
		PRINT("\nElements in the circular buffer:\r\n");

		for((temp = (buffer->front_CB)); (temp!=(buffer->rear_CB));(temp= (temp-
>link)))
		{
			#ifdef FRDM
			send_to_console((temp->data));
			#else
			printf("%d", temp->data);
			#endif
			PRINT("\r\n");
			count ++;



		}

		//print last eleemnt of the buffer
		#ifdef FRDM
		send_to_console(((buffer->rear_CB)->data));
		#else
		printf("%d",((buffer->rear_CB)->data));
		#endif

		PRINT("\r\n");

		//print the count of elements
```

```
#ifdef FRDM
char count_str[30];
sprintf(count_str,"Total elements present = %lu \r\n",(count+1));
PRINT(count_str);
#else
printf("Total elements present = %u \n",(count+1));
#endif

return SUCCESS;
	}
}


//*************************************************************************
// Name        : pop_data
//
// Description : Function to pop data out of circular Buffer
//
// Author      : Steve and Swarupa
//
// Date        : Nov 18, 2018
//
// Arguments   : No
//
// return      : status
//                          SUCCESS on completion, otherwise ERROR
//
//*************************************************************************/


//*****************************************************************************
// Include files
//*****************************************************************************
#include "pop_data.h"

//*****************************************************************************
// Function definition
//*****************************************************************************

status pop_data(CB *buffer,uint8_t *data)
{

	//checks if it is a null pointer
	if(buffer==NULL)
	{
		PRINT("Null pointer input\r\n");
		return NULL_PTR;
	}

	*data = 0;

	//check if the buffer is initialised
	if((buffer->flag_init) != 1)
	{
		PRINT("Buffer not initialized\r\n");
```

```
		return BUFFER_NOT_INITIALISED;
	}

	//check if the buffer is empty
	else if(IsEMPTY(buffer))
	{
		PRINT("Nothing to pop: Buffer is empty\r\n");
		return EMPTY;

	}

	//pop data
	else
	{
		(*data) = ((buffer->front_CB) -> data);
		(buffer->front_CB) = ((buffer->front_CB) -> link);
		(buffer->length_CB)--;
		return SUCCESS;
	}
}

//*************************************************************************
// Name        : insert_link.c
//
// Description : Function to insert a link to the circular linked list
//
// Author      : Steve and Swarupa
//
// Date        : Nov 06, 2018
//
// Arguments   : No
//
// return      : unused
//
//*************************************************************************/

//*************************************************************************************
// Include files
//*************************************************************************************
#include "insert_link.h"

//*************************************************************************************
// Function definition
//*************************************************************************************

status insert_link(CB *buffer)
{
    //checks if it is a null pointer
    if(buffer==NULL)
    {
        PRINT("Null pointer input\r\n");
        return NULL_PTR;
    }

    //creating a node
```

```
      struct node *temp = (struct node *) malloc (sizeof(struct node));
    if (temp == NULL)
    {
        PRINT("no memory space\r\n");
        return ERROR;
    }
    temp->data = '\0';
    temp->link = NULL;


    //for the first node
    if((buffer->head)!= NULL)
    {
       (temp->link) = (buffer->head);
        (buffer->head) = temp;
        ((buffer->tail)->link) = (buffer->head);
    }

    //for nodes other than the first node
    else
    {
       (buffer->head) = temp;
       (buffer->tail) = temp;
              ((buffer->tail) -> link) = temp;
    }
    (buffer->max_size)++;

    return SUCCESS;

}


//*************************************************************************
// Name        : insert_data
//
// Description : Function to insert data to the circular Buffer
//
// Author      : Steve and Swarupa
//
// Date        : Nov 06, 2018
//
// Arguments   : val - Data to be added
//
// return      : status
//                          SUCCESS on completion, otherwise ERROR
//
//*************************************************************************/


//*************************************************************************************
// Include files
//*************************************************************************************
#include "insert_data.h"

//*************************************************************************************
```

```c
// Function definition
//*****************************************************************************

status insert_data(CB *buffer,uint8_t val)
{
	//checks if it is a null pointer
	if(buffer==NULL)
    {
	 PRINT("Null pointer input\r\n");
	 return NULL_PTR;
    }

	//check for buffer initialisation
	if((buffer->flag_init) != 1)
	{
		PRINT("Buffer not initialized\r\n");
		return BUFFER_NOT_INITIALISED;
	}

	//check for space in buffer
	else if(IsFULL(buffer))
	{
		PRINT("\nThe Buffer is FULL\r\n");
		PRINT("Adding failed\r\n");
		return OVERFLOW;
	}

	//adding elements in the buffer when it is added for the first time from empty
state
	else if(((buffer->front_CB) == (buffer->rear_CB))&& (IsEMPTY(buffer)))
	{
		((buffer->rear_CB) -> data) = val;
		(buffer->length_CB)++;
#ifdef DEBUG
		PRINT("Added to the buffer\r\n");
		send_to_console((buffer->rear_CB) -> data);
		PRINT("\r\n");
#endif
		return SUCCESS;
	}

	//adding elements to the buffer
	else
	{
		(buffer->rear_CB) = ((buffer->rear_CB) -> link);
		((buffer->rear_CB) -> data) = val;
		(buffer->length_CB)++;
#ifdef DEBUG
		PRINT("Added to the buffer\r\n");
		send_to_console((buffer->rear_CB) -> data);
		PRINT("\r\n");
#endif
		return SUCCESS;
	}
}
```

```c
/**
* @file init_CB.c
*
* This file contains function for checking if the circular buffer is empty, full
* and initiating the circular buffer
*
* @author Steve and Swarupa
* @date Nov 6, 2018
*
*/

//*****************************************************************************
// Include files
//*****************************************************************************
#include "init_CB.h"

//*****************************************************************************
// Function definition
//*****************************************************************************

//*****************************************************************************
// Name        : init_CB
//
// Description : Function to initiate a circular buffer
//
// Arguments   : length - Length of the circular buffer to be created
//
// return      : SUCCESS if it is created successfully, otherwise ERROR
//
//*****************************************************************************/


status init_CB(CB *buffer,int32_t length)
{
       //checks if it is a null pointer
       if(buffer==NULL)
    {
        PRINT("Null pointer input\r\n");
        return NULL_PTR;
    }

       //check if the length is valid
       if(length<=0)
       {
              PRINT("Not a valid length\r\n");
              return ERROR;
       }

       (buffer->head) = NULL;
       (buffer->tail) = NULL;
       (buffer->front_CB) = NULL;
       (buffer->rear_CB) = NULL;
       (buffer->length_CB) = 0;
       (buffer->max_size) = 0;
```

```
        (buffer->flag_init) = 0;



        //creating the circular buffer

                    //flag for buffer initialisation
              (buffer->flag_init) = 1;

              //circular linked list creation
              for(int i=1; i<=length;i++)
              {
                    if((insert_link(buffer)) == ERROR)
                    {
                          PRINT("No space to allocate\r\n");
                          return ERROR;
                    }
         }



    (buffer->front_CB) = (buffer->head);
    (buffer->rear_CB) = (buffer->head);

        return SUCCESS;
}

//*****************************************************************************
// Name        : IsFULL
//
// Description : Function to check if the circular buffer is FULL
//
// Arguments   : No
//
// return      : 1 if it is Full, otherwise 0
//
//*****************************************************************************/

int8_t IsFULL(CB *buffer)
{
        if((buffer->length_CB) == (buffer->max_size))
              return 1;
        else
              return 0;
}


//*****************************************************************************
// Name        : IsEMPTY
//
// Description : Function to check if the circular buffer is Empty
//
// Arguments   : No
//
// return      : 1 if it is Empty, otherwise 0
```

```
//
//*****************************************************************************/
int8_t IsEMPTY(CB *buffer)
{
	if((buffer->length_CB) == 0)
		return 1;
	else
		return 0;
}



//*****************************************************************************
// Name		: delete_CB
//
// Description : Function to delete data from the circular Buffer
//
// Author	: Steve and Swarupa
//
// Date		: Nov 06, 2018
//
// Arguments	: No
//
// return	: status
//				SUCCESS on completion, otherwise ERROR
//
//*****************************************************************************/


//*****************************************************************************
// Include files
//*****************************************************************************
#include "delete_CB.h"

//*****************************************************************************
// Function definition
//*****************************************************************************

status delete_CB(CB *buffer)
{
	//checks if it is a null pointer
	if(buffer==NULL)
    {
	  PRINT("Null pointer input\r\n");
	  return NULL_PTR;
    }

	//check if the buffer is initialised
	if((buffer->flag_init) != 1)
	{
		PRINT("Buffer not initialized\r\n");
		return BUFFER_NOT_INITIALISED;
	}

	//desroying the circular buffer
```

```
	// Traversing through the circular linked list and deleting
	else
	{
		// creating a temporary variable
		CB *tmp;
		CB tmp_1;
		tmp = &tmp_1;

		//freeing the circular linked list
		while((buffer->max_size)>0)
		{
			#ifdef DEBUG
			printf("max size %d\n",(buffer->max_size));
			printf("head %p\n",(buffer->head));

			#endif

			//traversing through the linked list
			(tmp->head) = (buffer->head);
			(buffer->head) = ((buffer->head) -> link);
			free((tmp->head));
			(buffer->max_size)--;
		}

		//making all pointers NULL after freeing
		(buffer->head) = NULL;
		(buffer->tail) = NULL;
		(buffer->front_CB) = NULL;
		(buffer->rear_CB) = NULL;
		(buffer->length_CB) = 0;
		(buffer->max_size) = 0;
		(buffer->flag_init) = 0;
		PRINT("Deleted successfully\r\n");
		return SUCCESS;
	}
}

//*****************************************************************************
// Name        : clear_buffer
//
// Description : Function to clear the data present in circular buffer
//
// Author      : Steve and Swarupa
//
// Date        : Nov 07, 2018
//
// Arguments   : No
//
// return      : status
//                         SUCCESS on completion, otherwise ERROR
//
//*****************************************************************************/


//*****************************************************************************
```

```c
// Include files
//*****************************************************************************
#include "clear_buffer.h"

//*****************************************************************************
// Function definition
//*****************************************************************************

status clear_buffer(CB *buffer)
{
	//checks if it is a null pointer
	if(buffer==NULL)
    {
	  PRINT("Null pointer input\r\n");
	  return NULL_PTR;
    }

	//Check if the buffer is not initialized
	if((buffer->flag_init) != 1)
	{
		PRINT("Buffer not initialized\r\n");
		return BUFFER_NOT_INITIALISED;
	}

	//check if the buffer is empty
	else if(IsEMPTY(buffer))
	{
		PRINT("Nothing to clear: Buffer is empty\r\n");
		return EMPTY;
	}

	//clearing the buffer contents
	else
	{
		(buffer->front_CB) = (buffer->head);
		(buffer->rear_CB) = (buffer->head);
		(buffer->length_CB) = 0;
		PRINT("Cleared the buffer\r\n");
		return SUCCESS;
	}
}

/**
* @file main.c
*
*
* @author Steve and Swarupa
* @date Nov 6, 2018
*
*/

//*****************************************************************************
// Include files
//*****************************************************************************
```

```c
#include "main.h"


//*****************************************************************************
// Globals
//*****************************************************************************

uint32_t database[256] = {0};
uint8_t data_pop = 0;
char num[20];
uint32_t prime_number;
char prime_print[30];
//variable to store the received data
uint8_t data_poll = 0;

//Flag to indicate that a data is received
int8_t FLAG_RECV = 0;

//*****************************************************************************
// Function definition
//*****************************************************************************

//*****************************************************************************
// Name        : main
//
// Description : main function
//
// Arguments   : none
//
// return      : unused
//
//*****************************************************************************/


int main(void)
{


#ifdef LINUX
	init_CB(&RX_buffer,5);

	delete_CB(&RX_buffer);
	delete_CB(&RX_buffer);

	report_data(&RX_buffer);


#endif

#ifdef FRDM


	//Interrupt mode - Character Histogram application
#ifdef INTERRUPT
```

```
	//initiating UART0
	uartinit();

	//Initiating UART Rx interrupt
	RX_interrupt_init();

	//Initiating LED Red
	LED_init_IRQ();

	send_to_console_str("Welcome to Character Histogram Application\r\n");

	//initiating the circular buffer
	init_CB(&RX_buffer, SIZE_OF_RX_CB);

	PTB->PSOR |= (1<<18);//off LED red

	while(1)
	{

	    uint32_t value,divisor,Exact_divisible_count;

	    //Prime number generator
	    for(value=2;value<MAX_PRIME_NUMBER;value++)
	       {
	       Exact_divisible_count =0;

	              for(divisor=1;divisor<=value;divisor++)
	              {
	                     //Checks if the remainder is zero
	                     if(value%divisor == 0)
	                     {
	                            Exact_divisible_count++;
	                     }
	              }

	              //If Exact_divisible_count=2, it is a prime number
	              if(Exact_divisible_count==2)
	              {
	                  prime_number = value ;
	              }

	       }
	}

#else

	//polling mode - echoing characters


	//configuring interrupt
	uartinit();
```

```
	while(1)
	{
		// Receiver polling
		while(UART0_S1 & UART0_S1_RDRF_MASK)
		{
			receiver_polling();
		}

		//Transmitting
		while(FLAG_RECV == 1)
		{
			while(UART0_S1 & UART0_S1_TDRE_MASK)
			{
				transmitter_polling();
			}
			FLAG_RECV = 0;
		}

	}
#endif
#endif

	return 1;
}

#ifdef FRDM
//*****************************************************************************
// Function definition
//*****************************************************************************

//*****************************************************************************
// Name        : sys_reload
//
// Description : function to reload the systick timer
//
// Arguments   : none
//
// return      : unused
//
//*****************************************************************************/
void sys_reload()
{
	SYSTICK_VAL = 0x0;    //clear current timer value
	SYSTICK_LOAD = 0xFFFFFFF; //loading value
	SYSTICK_CTRL = 0x7; //enabling systick interrupt
}

//*****************************************************************************
// Systick ISR definition
//*****************************************************************************

//*****************************************************************************
// Name        : SysTick_Handler
//
// Description : ISR for systick timer which has the report generation part
```

```
//
// Arguments    : none
//
// return       : unused
//
//*****************************************************************************/
void SysTick_Handler(void)
{
	PTB->PCOR = (1<<18); //on red LED

	PRINT("\r\n\r\n");
	PRINT("Report:\r\n");

	//Report Generation Part
	for(int i =0;i<MAX_ASCII;i++)
	{
		if(database[i]!= 0)
		{
			sprintf(num," %c - %lu \r\n",i,database[i]);
			PRINT(num);
		}
	}


	//Triangle pattern generation using prime number

	sprintf(prime_print,"%lu ",prime_number);

	PRINT("--------------------------------\r\n");
	//Pattern
	for(int row=1; row<=ROWS_PATTERN_MAX; row++)
	{
	     for(int column=1; column<=row; column++)
		 {
			PRINT(prime_print);
		 }
		 PRINT("\r\n");
	}

	// disabling the systick interrupt
	SYSTICK_CTRL = 0x0;

	PTB->PSOR = (1<<18); // off red LED
}
//*****************************************************************************
// Name         : LED_init_IRQ
//
// Description : For initiating LED
//
// Arguments    : none
//
// return       : unused
//
//*****************************************************************************/
void LED_init_IRQ()
```

```
{
	//Clock for PORT B
	SIM_BASE_PTR->SCGC5 |= SIM_SCGC5_PORTB_MASK;

	//Alternate function selection for PIN 18 of PORT B
	PORTB_PCR18 |= PORT_PCR_MUX(0x1);

	PTB->PDDR |= (1<<18); // selecting as output pin

}
#endif

/*****************************************************************************
 * Filename: Unittest1.c
 * Author : Steve and Swarupa
 *
 * Description : This file tests each and every functionality used individually using
CUnits
 *
 * ***************************************************************************/

 /*************************************************************/
 //    Includes
 /*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "../inc/common/common_variable.h"
#include "CUnit/Basic.h"
#include "../inc/common/main.h"
#include <time.h>

 /*************************************************************/
 //    MACROS
 /*************************************************************/

#define MAX_DATA_VALUE (255) // largest value of ascii
#define MAX_ARRAY_LENGTH (1000)
#define MIN_ARRAY_LENGTH (10)

 /*************************************************************/
 //    GLOBALS
 /*************************************************************/
uint8_t *p = NULL;
uint8_t data_pop= 0;
uint8_t parameter_value[2000] = {0};
CB buffer_1;
CB buffer_2;
CB buffer_3;
CB buffer_4;
CB buffer_5;
CB buffer_6;
CB buffer_7;
CB buffer_8;
```

```
int32_t CB_SIZE = 0;


//*****************************************************************************
// Function definition
//*****************************************************************************

//*****************************************************************************
// Name        : random_generator
//
// Description : Generate an array of random number for giving as input in various
functions
//
// Arguments   : none
//
// return      : unused
//
//*****************************************************************************/

/*random value generator function */
int random_generator()
{

  for(uint32_t i= 0; i< CB_SIZE; i++)
    {
     //limiting the random value to be within max value of ASCII
     parameter_value[i] = (rand() % MAX_DATA_VALUE );
    }
    return 0;

}

//*****************************************************************************
// initiating suite
//*****************************************************************************

/*Suite 1*/
int init_suite_init_CB(void)
{
      return 0;
}

/*Suite 2*/
int init(void)
{
      init_CB(&buffer_2, (CB_SIZE));
      init_CB(&buffer_3, (CB_SIZE));
      return 0;

}

/*Suite 3*/
int init_delete(void)
{
      init_CB(&buffer_4, (CB_SIZE));
```

```
      return 0;
}

/*Suite 4*/
int init_suite_report_data(void)
{
    init_CB(&buffer_5, (CB_SIZE));
      return 0;
}

/*Suite 5*/

int init_suite_resize(void)
{
    init_CB(&buffer_6, (CB_SIZE));
    init_CB(&buffer_7, (CB_SIZE));

      return 0;
}

/*Suite 6*/
int init_suite_clear(void)
{
    init_CB(&buffer_8, (CB_SIZE));
     uint32_t i;

      for(p= parameter_value,i=0; i<CB_SIZE ; i++, p++)
      {
            insert_data(&buffer_8,*p);
      }
      return 0;
}

//*************************************************************************************
// Cleaning suite
//*************************************************************************************

int clean_suite(void)
{
      return 0;
}

//*************************************************************************************
// Adding test registry
//*************************************************************************************

void test_init_CB() //suite1
{

     p= parameter_value;

     CB *circular_buffer;
     circular_buffer = NULL;

     //NULL pointer check
```

```
    CU_ASSERT_EQUAL(init_CB(circular_buffer, 5), NULL_PTR);
    CU_ASSERT_EQUAL(insert_data(circular_buffer,*p),NULL_PTR);
       CU_ASSERT_EQUAL(delete_CB(circular_buffer),NULL_PTR);
       CU_ASSERT_EQUAL(report_data(circular_buffer),NULL_PTR);
       CU_ASSERT_EQUAL(clear_buffer(circular_buffer),NULL_PTR);
       CU_ASSERT_EQUAL(resize_CB(circular_buffer,*p),NULL_PTR);
       CU_ASSERT_EQUAL(pop_data(circular_buffer,&data_pop),NULL_PTR);

       //Check when the Circular buffer functions are called without initializing the
circular buffer
       CU_ASSERT_EQUAL(insert_data(&buffer_1,*p),BUFFER_NOT_INITIALISED);
       CU_ASSERT_EQUAL(delete_CB(&buffer_1),BUFFER_NOT_INITIALISED);
       CU_ASSERT_EQUAL(report_data(&buffer_1),BUFFER_NOT_INITIALISED);
       CU_ASSERT_EQUAL(clear_buffer(&buffer_1),BUFFER_NOT_INITIALISED);
       CU_ASSERT_EQUAL(resize_CB(&buffer_1,*p),BUFFER_NOT_INITIALISED);
       CU_ASSERT_EQUAL(pop_data(&buffer_1,&data_pop),BUFFER_NOT_INITIALISED);


       CU_ASSERT_EQUAL(init_CB(&buffer_1, -1), ERROR);
       CU_ASSERT_EQUAL(init_CB(&buffer_1, 0), ERROR);
       CU_ASSERT_EQUAL(init_CB(&buffer_1, 1), SUCCESS);

       CU_ASSERT_EQUAL(init_CB(&buffer_1,*p),SUCCESS);


       CU_ASSERT_EQUAL(delete_CB(&buffer_1), SUCCESS);

}

void test_insert_data() //suite2
{
       uint32_t i;

       //inserting elements
    for(p= parameter_value,i=0; i<(CB_SIZE) ; i++, p++)
       {
              CU_ASSERT_EQUAL(insert_data(&buffer_2,*p),SUCCESS);

       }

       //Buffer overflow
       for(p= parameter_value,i=0; i<(CB_SIZE) ; i++, p++)
       {
                     CU_ASSERT_EQUAL(insert_data(&buffer_2,*p),-4);

       }

       //boundary conditions
       CU_ASSERT_EQUAL(insert_data(&buffer_3, '0'),SUCCESS);
       CU_ASSERT_EQUAL(insert_data(&buffer_3, '&'),SUCCESS);
       CU_ASSERT_EQUAL(insert_data(&buffer_3, ','),SUCCESS);
       CU_ASSERT_EQUAL(insert_data(&buffer_3, '\n'),SUCCESS);
       CU_ASSERT_EQUAL(insert_data(&buffer_3, ' '),SUCCESS);
       CU_ASSERT_EQUAL(insert_data(&buffer_3, 'z'),SUCCESS);
       CU_ASSERT_EQUAL(insert_data(&buffer_3, '\\'),SUCCESS);
```

```
	CU_ASSERT_EQUAL(insert_data(&buffer_3, 5),SUCCESS);
	CU_ASSERT_EQUAL(insert_data(&buffer_3, 0),SUCCESS);



	CU_ASSERT_EQUAL(delete_CB(&buffer_2), SUCCESS);
	CU_ASSERT_EQUAL(delete_CB(&buffer_3), SUCCESS);



}

void test_delete_data() //suite3
{

	uint32_t i;
	for(i=0; i<CB_SIZE ; i++)
	{
		if(i == 0)
		{
			CU_ASSERT_EQUAL(delete_CB(&buffer_4),SUCCESS);
		}
		else
		{
			CU_ASSERT_EQUAL(delete_CB(&buffer_4),BUFFER_NOT_INITIALISED);
		}
	}

}

void test_pop_and_report_data() //suite4
{
	uint32_t i;

	//inserting elements - success
	for(p= parameter_value,i=0; i<(CB_SIZE) ; i++, p++)
	{
		CU_ASSERT_EQUAL(insert_data(&buffer_5,*p),SUCCESS);
	}

	//report data - success
	for(i=0; i<(CB_SIZE) ; i++)
	{

		CU_ASSERT_EQUAL(report_data(&buffer_5),SUCCESS);
	}

	//pop data - success
	for(i=0; i<(CB_SIZE) ; i++)
	{

			CU_ASSERT_EQUAL(pop_data(&buffer_5, &data_pop),SUCCESS);
	}

	//report data - empty
```

```
	for(i=0; i<(CB_SIZE) ; i++)
	{

		CU_ASSERT_EQUAL(report_data(&buffer_5),EMPTY);
	}

	//pop data - empty
	for(i=0; i<(CB_SIZE) ; i++)
	{

			CU_ASSERT_EQUAL(pop_data(&buffer_5, &data_pop),EMPTY);
	}


	CU_ASSERT_EQUAL(delete_CB(&buffer_5),SUCCESS);
}

void test_resize_CB() //suite5
{
	uint32_t i;

	//insert data - success
     for(p= parameter_value,i=0; i<(CB_SIZE) ; i++, p++)
	{
		CU_ASSERT_EQUAL(insert_data(&buffer_6,*p),SUCCESS);
	}

	//insert data - overflow
	CU_ASSERT_EQUAL(insert_data(&buffer_6,parameter_value[0]),-4);

	//resize
	CU_ASSERT_EQUAL(resize_CB(&buffer_6,CB_SIZE),SUCCESS);

	//insert data - success
	 for(p= parameter_value,i=0; i<(CB_SIZE) ; i++, p++)
	{
		CU_ASSERT_EQUAL(insert_data(&buffer_6,*p),SUCCESS);
	}

	//boundary cases
	CU_ASSERT_EQUAL(resize_CB(&buffer_7,-2),ERROR);
	CU_ASSERT_EQUAL(resize_CB(&buffer_7,0),ERROR);


	CU_ASSERT_EQUAL(delete_CB(&buffer_6),SUCCESS);
	CU_ASSERT_EQUAL(delete_CB(&buffer_7),SUCCESS);



}

void test_clear_buffer() //suite5
{

   CU_ASSERT_EQUAL(clear_buffer(&buffer_8),SUCCESS);
```

```
   CU_ASSERT_EQUAL(clear_buffer(&buffer_8),EMPTY);

}

//*************************************************************************
// Name        : main function
//
// Description : contains the call of the suites for unit testing
//
// Arguments   : none
//
// return      : unused
//
//*************************************************************************/
int main(void)
{

        if (CUE_SUCCESS != CU_initialize_registry())
       return CU_get_error();

       CU_pSuite pSuite1 = NULL;
    /* initialize the CUnit test registry */


       /* add a suite to the registry */
       pSuite1 = CU_add_suite("Suite_1", init_suite_init_CB, clean_suite);
    if (NULL == pSuite1)
    {
            CU_cleanup_registry();
        return CU_get_error();
       }

    /* add the tests to the suite */
    /* NOTE - ORDER IS IMPORTANT - MUST TEST fread() AFTER fprintf() */
    if ((NULL == CU_add_test(pSuite1, "test of fprintf()", test_init_CB)))
    {
        CU_cleanup_registry();
       return CU_get_error();
       }
/*************************************************************************/

       CU_pSuite pSuite2 = NULL;

       /* add a suite to the registry */
       pSuite2 = CU_add_suite("Suite_2", init, clean_suite);
    if (NULL == pSuite2)
    {
            CU_cleanup_registry();
        return CU_get_error();
       }


       if ((NULL == CU_add_test(pSuite2, "test of fprintf()", test_insert_data)))
    {
        CU_cleanup_registry();
```

```
        return CU_get_error();
    }

/************************************************************************/

      CU_pSuite pSuite3 = NULL;


      /* add a suite to the registry */
      pSuite3 = CU_add_suite("Suite_3", init_delete, clean_suite);
    if (NULL == pSuite3)
    {
           CU_cleanup_registry();
        return CU_get_error();
      }

      if ((NULL == CU_add_test(pSuite3, "test of fprintf()", test_delete_data)))
      {
          CU_cleanup_registry();
          return CU_get_error();
      }
/************************************************************************/

      CU_pSuite pSuite4 = NULL;


      /* add a suite to the registry */
      pSuite4 = CU_add_suite("Suite_4", init_suite_report_data, clean_suite);
    if (NULL == pSuite4)
    {
           CU_cleanup_registry();
        return CU_get_error();
      }
      if ((NULL == CU_add_test(pSuite4, "test of fprintf()",
test_pop_and_report_data)))
      {
            CU_cleanup_registry();
            return CU_get_error();
      }
/************************************************************************/

      CU_pSuite pSuite5 = NULL;


      /* add a suite to the registry */
      pSuite5 = CU_add_suite("Suite_5", init_suite_resize, clean_suite);
    if (NULL == pSuite5)
    {
           CU_cleanup_registry();
        return CU_get_error();
      }

      if ((NULL == CU_add_test(pSuite5, "test of fprintf()", test_resize_CB)))
      {
            CU_cleanup_registry();
```

```
                return CU_get_error();
        }

/************************************************************************/

        CU_pSuite pSuite6 = NULL;


        /* add a suite to the registry */
        pSuite6 = CU_add_suite("Suite_6", init_suite_clear, clean_suite);
    if (NULL == pSuite6)
    {
                CU_cleanup_registry();
            return CU_get_error();
        }
        if ((NULL == CU_add_test(pSuite6, "test of fprintf()", test_clear_buffer)))
        {
                CU_cleanup_registry();
                return CU_get_error();
        }

/************************************************************************/
    /* Run all tests using the CUnit Basic interface */
    time_t t;
    /* Intializes random number generator */
   srand((unsigned) time(&t));

  //parameter generation for circular buffer length

  refetch:
    CB_SIZE = ((rand()) % MAX_ARRAY_LENGTH);

    if((CB_SIZE < MIN_ARRAY_LENGTH))
    {
      goto refetch;
    }

    printf("CB Length  %d\n",CB_SIZE);
    random_generator();
      CU_basic_set_mode(CU_BRM_VERBOSE);
      CU_basic_run_tests();
      CU_cleanup_registry();
return CU_get_error();
}
```