MILESTONE 3 - Linked Lists and MCU I/O

February 5, 2024

MECH 458 - B01 Group 1

Mckinlay, Samantha (V00954147)

Berezowska, Kiara (V00937549)

Documented Codes

```
/* Solution Set for the LinkedQueue.c */
/*
        Course          : UVic Mechatronics 458
        Milestone       : 3
        Title           : Data structures for MCUs and the Linked Queue Library

        Name 1:     Mckinlay, Samantha          Student ID: V00954147
        Name 2:     Berezowska, Kiara           Student ID: V00937549

        Description: Use mTimer, Linked Lists, User Input, and button to display items in list shifted two bits each time.
*/

/* include libraries */
#include <stdlib.h>
#include <avr/io.h>
#include "LinkedQueue.h"        /* This is the attached header file, which cleans things up */

void mTimer(int count);
int main(int argc, char *argv[]){
```

```
CLKPR = 0x80;
CLKPR = 0x01;

TCCR1B = _BV(CS11);

char readInput;

link *head;                     /* The ptr to the head of the queue */
link *tail;                     /* The ptr to the tail of the queue */
link *newLink;          /* A ptr to a link aggregate data type (struct) */
link *rtnLink;          /* same as the above */
element eTest;          /* A variable to hold the aggregate data type known as element */

DDRC = 0xFF;            /* Used for debugging purposes only LEDs on PORTC */
//DDRD = 0xFF;
DDRA = 0x00; // sets port A to input
DDRC = 0xFF; //sets port C to output bits for red LEDs

while(1){
        rtnLink = NULL;
        newLink = NULL;
        setup(&head, &tail);
        for(int i = 0; i<3; i++){
                //check button
                while((PINA&0x04)==0x04); //checking if button is HIGH ie pushed
                mTimer(20); //debounce
                initLink(&newLink);
                newLink->e.itemCode = (PINA&0x03);
                enqueue(&head, &tail, &newLink);
                //check button
                while((PINA&0x04)==0x00); //checking if button is LOW ie not pushed
```

```
                mTimer(20); //debounce
            }
            int i = 0;
            while(isEmpty(&head)!=1){
                dequeue(&head, &rtnLink); //remove the item at the head of the list
                PORTC = rtnLink->e.itemCode<<i;
                mTimer(2000);
                i= i+2;
            }

            while((PINA&0x04)==0x04);
            mTimer(20); //debounce
            PORTC = 0;
            while((PINA&0x04)==0x00); //checking if button is LOW ie not pushed
            mTimer(20); //debounce
        }
        return(0);
}/* main */


/*****************************************************************************
**/
/***************************** SUBROUTINES
*****************************************/
/*****************************************************************************
**/



/*****************************************************************************
**
* DESC: initializes the linked queue to 'NULL' status
* INPUT: the head and tail pointers by reference
*/
```

```
void setup(link **h,link **t){
        *h = NULL;              /* Point the head to NOTHING (NULL) */
        *t = NULL;              /* Point the tail to NOTHING (NULL) */
        return;
}/*setup*/
```

```
/*****************************************************************************
**
* DESC: This initializes a link and returns the pointer to the new link or NULL if error
* INPUT: the head and tail pointers by reference
*/
void initLink(link **newLink){
        //link *l;
        *newLink = malloc(sizeof(link));
        (*newLink)->next = NULL;
        return;
}/*initLink*/
```

```
/*****************************************************************************
****
*  DESC: Accepts as input a new link by reference, and assigns the head and tail
*  of the queue accordingly
*  INPUT: the head and tail pointers, and a pointer to the new link that was created
*/
/* will put an item at the tail of the queue */
```

```
void enqueue(link **h, link **t, link **nL){

	if (*t != NULL){
		/* Not an empty queue */
		(*t)->next = *nL;
		*t = *nL; //(*t)->next;
	}/*if*/
	else{
		/* It's an empty Queue */
		//(*h)->next = *nL;
		//should be this
		*h = *nL;
		*t = *nL;
	}/* else */
	return;
}/*enqueue*/




/*****************************************************************************
**
* DESC : Removes the link from the head of the list and assigns it to deQueuedLink
* INPUT: The head and tail pointers, and a ptr 'deQueuedLink'
*                  which the removed link will be assigned to
*/
/* This will remove the link and element within the link from the head of the queue */
void dequeue(link **h, link **deQueuedLink){
	/* ENTER YOUR CODE HERE */
	*deQueuedLink = *h;    // Will set to NULL if Head points to NULL
	/* Ensure it is not an empty queue */
	if (*h != NULL){
```

```
                *h = (*h)->next;
        }/*if*/

        return;
}/*dequeue*/
```

```
/*****************************************************************************
**
* DESC: Peeks at the first element in the list
* INPUT: The head pointer
* RETURNS: The element contained within the queue
*/
/* This simply allows you to peek at the head element of the queue and returns a NULL pointer if empty
*/
element firstValue(link **h){
        return((*h)->e);
}/*firstValue*/
```

```
/*****************************************************************************
**
* DESC: deallocates (frees) all the memory consumed by the Queue
* INPUT: the pointers to the head and the tail
*/
/* This clears the queue */
void clearQueue(link **h, link **t){
```

```
	link *temp;

	while (*h != NULL){
		temp = *h;
		*h=(*h)->next;
		free(temp);
	}/*while*/

	/* Last but not least set the tail to NULL */
	*t = NULL;

	return;
}/*clearQueue*/
```

```
/*****************************************************************************
**
* DESC: Checks to see whether the queue is empty or not
* INPUT: The head pointer
* RETURNS: 1:if the queue is empty, and 0:if the queue is NOT empty
*/
/* Check to see if the queue is empty */
char isEmpty(link **h){
	/* ENTER YOUR CODE HERE */
	return(*h == NULL);
}/*isEmpty*/
```

```
/*****************************************************************************
**
* DESC: Obtains the number of links in the queue
* INPUT: The head and tail pointer
* RETURNS: An integer with the number of links in the queue
*/
/* returns the size of the queue*/
int size(link **h, link **t){

        link    *temp;                  /* will store the link while traversing the queue */
        int     numElements;

        numElements = 0;

        temp = *h;                      /* point to the first item in the list */

        while(temp != NULL){
                numElements++;
                temp = temp->next;
        }/*while*/

        return(numElements);
}/*size*/

/*****************************************************************************
**
* DESC: Acts as a clock.
* INPUT: Amount of time that has to be counted.
* RETURNS: Nothing
```

```
*/
void mTimer (int count){
  /***
    Setup Timer1 as a ms timer
          Using polling method not Interrupt Driven
  ***/

  int i;

  i = 0;

  //TCCR1B |= _BV (CS11);  // Set prescaler (/8) clock 16MHz/8 -> 2MHz
  /* Set the Waveform gen. mode bit description to clear
    on compare mode only */
  TCCR1B |= _BV(WGM12);

  /* Set output compare register for 1000 cycles, 1ms */
  OCR1A = 0x03E8;

  /* Initialize Timer1 to zero */
  TCNT1 = 0x0000;

  /* Enable the output compare interrupt */
  TIMSK1 = TIMSK1 | 0b00000010;

  /* Clear the Timer1 interrupt flag and begin timing */
  TIFR1 |= _BV(OCF1A);

  /* Poll the timer to determine when the timer has reached 1ms */
  while (i < count){
    if((TIFR1 & 0x02) == 0x02){
```

```
            /* Clear the interrupt flag by WRITING a ONE to the bit */
            TIFR1 |= _BV(OCF1A);
            i++;
            }
          }
  return;
}  /* mTimer */
```

Lab Assignment Solutions

No lab assignments.

Supplementary Information

No supplementary information.