```c
/*
 * MECH 458 - Lab 3
 * Benjamin Klammer - V00829094
 * Joshua Columbus - V00825727
 */

#include <asf.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "LinkedQueue.h"

void mTimer(int count);
int display(char input);

int main (int argc, char *argv[])
{
    CLKPR = 0b10000000; /* Modify prescale lock bit */
    CLKPR = 0b00000000; /* Reset prescaler to have 8MHz clock */

    TCCR1B |= _BV(CS10);/* Set timer/counter 1B */
    DDRA = 0b00000000;  /* Set all of Pin A to input bits */
    DDRC = 0b11111111;  /* Set all of Port C to output bits */
    PORTC = 0b00000000; /* Set all Port C to low (turn off LEDs) */

    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 */

    rtnLink = NULL;
    newLink = NULL;

    while (1) {
        setup(&head, &tail);
        PORTC = 0b00000000;
        for (int i=0; i<4; i++){
            while((PINA&0b00000100) == 0b00000100){}    /* Test if button is
              pressed */
            mTimer(20);                                 /* Debouncer */
            initLink(&newLink);                         /* Initialize a space in
              memory and point to it with newLink */
            newLink->e.itemCode = PINA & 0b00000011;    /* Mask and save PINA
              state in a link */
            enqueue(&head, &tail, &newLink);            /* Enqueue PINA state */
            while((PINA&0b00000100) == 0b00000000){}    /* Test if button has
              stopped being pressed */
            mTimer(20);                                 /* Debouncer */
        } /*for */
```

```
        //Display lights
        dequeue(&head, &rtnLink);           /* Dequeue first link */
        free(rtnLink);                      /* Delete first link */

        dequeue(&head, &rtnLink);           /* Dequeue second link */
        PORTC += rtnLink->e.itemCode;       /* Output link to PORTC */
        free(rtnLink);                      /* Free link */
        mTimer(2000);                       /* Wait 2 seconds */

        dequeue(&head, &rtnLink);           /* Dequeue third link */
        PORTC += rtnLink->e.itemCode << 2;  /* Output link to PORTC */
        free(rtnLink);                      /* Free link */
        mTimer(2000);                       /* Wait 2 seconds */

        dequeue(&head, &rtnLink);           /* Dequeue fourth link */
        PORTC += rtnLink->e.itemCode << 4;  /* Output link to PORTC */
        free(rtnLink);                      /* Free link */
        mTimer(2000);                       /* Wait 2 seconds */

        while((PINA&0b00000100) == 0b00000100){}    /* Reset when pushed */
        mTimer(20);                                 /* Debouncer */
        while((PINA&0b00000100) == 0b00000000){}    /* Test if button has stopped ↵
          being pressed */
        mTimer(20);                                 /* Debouncer */
    } /* while */

    return(0);
}/* main */



/                                                                              ↵
  ****************************************************************************** ↵
  ******/
/*************************** SUBROUTINES                                        ↵
  ****************************************/
/                                                                              ↵
  ****************************************************************************** ↵
  ******/

/                                                                              ↵
  ****************************************************************************** ↵
  ******
* DESC: allows a set amount of time to elapse
* INPUT: the integer number to count to
*/
void mTimer (int count) {
    int i = 0;
    TCCR1B |= _BV(WGM12);
```

```
    OCR1A = 0x1F40;
    TCNT1 = 0x0000;
    TIMSK1 = TIMSK1|0b00000010;
    TIFR1 |= _BV(OCF1A);

    while (i<count) {
        if ((TIFR1 & 0x02) == 0x02) {
            TIFR1 |= _BV(OCF1A);
            i++;
        }
    }
    return;
}


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

```c
* 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*/
```