Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Klvana-Hooper	Student ID:	19782944
Other name(s):	Nicholas		
Unit name:	Operating Systems	Unit ID:	COMP2006
Lecturer / unit coordinator:	Sie Teng Soh	Tutor:	Sie Teng Soh
Date of submission:	16/05/2020	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature:  Date of signature: 16/05/2020

(By submitting this form, you indicate that you agree with all the above text.)

Part A

circularQueue.h

```
/*
 * File: CircularQueue.h
 * File Created: Tuesday, 5th May 2020
 * Author: Nicholas Klvana-Hooper
 * -----
 * Last Modified: Tuesday, 12th May 2020
 * Modified By: Nicholas Klvana-Hooper
 * -----
 * Purpose: Contains header information for CircularQueue.h
 * Reference:
 */
#ifndef CIRCULARQUEUE_H
#define CIRCULARQUEUE_H

/* Entry struct definition */
typedef struct{
   int start;
   int dest;
} Entry;

/* CircularQueue struct definition */
typedef struct{
   int max;
   int head;
   int tail;
   int count;
   int done;
   Entry* data;
} CircularQueue;
```

```
CircularQueue* createCircularQueue(int size);
int isEmpty(CircularQueue* queue);
int isFull(CircularQueue* queue);
void enqueue(CircularQueue* queue, Entry ent);
Entry* dequeue(CircularQueue* queue);
Entry* peek(CircularQueue* queue);
void freeQueue(CircularQueue* queue);
int isDone(CircularQueue* queue);
void setDone(CircularQueue* queue);

#endif
```

circularQueue.c

```
/*
 * File: circularQueue.c
 * File Created: Thursday, 7th May 2020
 * Author: Nicholas Klvana-Hooper
 * -----
 * Last Modified: Tuesday, 12th May 2020
 * Modified By: Nicholas Klvana-Hooper
 * -----
 * Purpose: Contains methods for a circular queue implementation
 * Reference:
 */
#include <stdio.h>
#include <stdlib.h>
#include "circularQueue.h"

/*
 * SUBMODULE: createCircularQueue
 * IMPORT: size(int)
 * EXPORT: queue (CircularQueue*)
```

```
 * ASSERTION: Creates and initialises the queue
 * REFERENCE:
 */
CircularQueue* createCircularQueue(int size)
{
   CircularQueue* queue;

   /* creates memory for and initialises the queue */
   queue = (CircularQueue*)malloc(sizeof(CircularQueue));
   queue->max = size;
   queue->head = 0;
   queue->tail = 0;
   queue->count = 0;
   queue->done = 0;
   queue->data = (Entry*)malloc(size * sizeof(Entry));

   return queue;
}

/*
 * SUBMODULE: isEmpty
 * IMPORT: queue (CircularQueue*)
 * EXPORT: bool(int)
 * ASSERTION: Returns an int representing a boolean saying if the queue is empty
 * REFERENCE:
 */
int isEmpty(CircularQueue* queue)
{
   int bool = 0;

   if(queue->count == 0)
   {
```

```
      bool = 1;
   }
   return bool;
}

/*
 * SUBMODULE: isFull
 * IMPORT: queue(CircularQueue*)
 * EXPORT: bool(int)
 * ASSERTION: Returns an int representing a boolean saying if the queue is full
 * REFERENCE:
 */
int isFull(CircularQueue* queue)
{
   int bool = 0;

   if(queue->count == queue->max)
   {
      bool = 1;
   }
   return bool;
}

/*
 * SUBMODULE: enqueue
 * IMPORT: queue(CircularQueue*), ent(Entry)
 * EXPORT: void
 * ASSERTION: Adds a new Entry object into the queue
 * REFERENCE:
 */
void enqueue(CircularQueue* queue, Entry ent)
{
```

```
   if(isFull(queue)==1)
   {
      printf("Error: Queue is full!\n");
   }
   else
   {
      /* perform the increment for circular queue tail */
      queue->data[queue->tail] = ent;
      queue->tail = (queue->tail + 1) % queue->max;
      queue->count += 1;
   }
}

/*
 * SUBMODULE: dequeue
 * IMPORT: queue(CircularQueue*)
 * EXPORT: ent(Entry*)
 * ASSERTION: Removes the next entry object from the queue
 * REFERENCE:
 */
Entry* dequeue(CircularQueue* queue)
{
   Entry* ent;
   Entry* blank;

   blank = (Entry*)malloc(sizeof(Entry));

   /* Create copy of next entry in queue */
   ent = peek(queue);

   /* Reduce count and head */
   queue->data[queue->head] = *blank;
```

```
    queue->head = (queue->head + 1) % queue->max;
    queue->count -= 1;

    free(blank);
    return ent;
}

/*
 * SUBMODULE: peek
 * IMPORT: queue(CircularQueue *)
 * EXPORT: ent (Entry*)
 * ASSERTION: Take a copy of the next queue in the queue
 * REFERENCE:
 */
Entry* peek(CircularQueue* queue)
{
    Entry* ent;

    ent = (Entry*)malloc(sizeof(Entry));

    if(isEmpty(queue) == 1)
    {
        printf("Error: Queue is empty!\n");
    }
    else
    {
        *ent = queue->data[queue->head];
    }
    return ent;
}

/*
```

```
 * SUBMODULE: freeQueue
 * IMPORT: queue(CircularQueue*)
 * EXPORT: void
 * ASSERTION: Frees all the segements and the queue itself
 * REFERENCE:
 */
void freeQueue(CircularQueue* queue)
{
   free(queue->data);
   free(queue);
}

/*
 * SUBMODULE: isDone
 * IMPORT: queue(CircularQueue*)
 * EXPORT: int
 * ASSERTION: Returns an int representing a boolean of whether the queue is done inputting
 * REFERENCE:
 */
int isDone(CircularQueue* queue)
{
   return queue->done;
}

/*
 * SUBMODULE: setDone
 * IMPORT: queue(CircularQueue*)
 * EXPORT: void
 * ASSERTION: Sets the done int within the queue
 * REFERENCE:
 */
void setDone(CircularQueue* queue)
```

```
{
   queue->done = 1;
}
```

Lift_sim_A.h

```
/*
 * File: lift_sim_A.h
 * File Created: Tuesday, 5th May 2020
 * Author: Nicholas Klvana-Hooper
 * -----
 * Last Modified: Tuesday, 12th May 2020
 * Modified By: Nicholas Klvana-Hooper
 * -----
 * Purpose: Contains header information for lift_sim_A.h
 * Reference:
 */

#ifndef LIFTSIMA_H
#define LIFTSIMA_H

void setup(char* argv[]);
void *request(void* vargp);
void *lift(void* vargp);

#endif
```

Lift_sim_A.c

```
/*
 * File: lift_sim_A.c
 * File Created: Thursday, 7th May 2020
 * Author: Nicholas Klvana-Hooper
 * -----
 * Last Modified: Tuesday, 12th May 2020
 * Modified By: Nicholas Klvana-Hooper
 * -----
 * Purpose: Runs a lift 3 lift simulator for a 20 story building
 * Reference:
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include "lift_sim_A.h"
#include "circularQueue.h"

/* Variables and data that is shared between threads */
CircularQueue* buffer;
FILE* output;
pthread_mutex_t lock, fileLock;
pthread_cond_t full, empty;
int sleepT, totReq, totalMove;

int main(int argc, char* argv[])
{
   if(argc != 3) /* needs buffer size and sleep*/
   {
      printf("Error! Usage should be ./lift_sim_A <bufferSize> <timeCount>\n");
   }
```

```
    else
    {
        if(sleepT >= 0) /* sleep must be 0 or greater */
        {
            sleepT = atoi(argv[2]);
            setup(argv);
        }
        else
        {
            printf("Error in sleep Variable! Must be 0 or bigger\n");
        }
    }

    return 0;
}

/*
 * SUBMODULE: setup
 * IMPORT: argv(char**)
 * EXPORT: void
 * ASSERTION: Splits into the 4 threads and runs the functions
 * REFERENCE:
 */
void setup(char* argv[])
{
    pthread_t r_id, l1_id, l2_id, l3_id;
    int l1, l2, l3, l_Count;
    char currChar;

    FILE* input = fopen("sim_input", "r");
    output = fopen("sim_output", "a");
    l1 = 1;
```

```
l2 = 2;
l3 = 3;
l_Count = 1;
currChar = 'a';
totalMove = 0;

/* initialises the mutex and conds*/
pthread_mutex_init(&lock, NULL);
pthread_mutex_init(&fileLock, NULL);
pthread_cond_init(&full, NULL);
pthread_cond_init(&empty, NULL);

/* buffer must be one or bigger */
if(atoi(argv[1]) >= 1)
{
   buffer = createCircularQueue(atoi(argv[1]));
}
else
{
   printf("Error: Buffer has to be 1 or larger!\n");
}

/* if input doesn't open */
if(input == NULL)
{
   perror("Error: could not open sim_input\n");
}
else if(ferror(input)) /* if error with input file */
{
   perror("Error reading from sim_input\n");
}
else
```

```
{
   /* loop through file counting number of lines */
   while(currChar != EOF)
   {
      currChar = fgetc(input);
      if(currChar == '\n')
      {
         l_Count++;
      }
   }
}

if(ferror(input))
{
   perror("Error when reading from sim_input\n");
}
else
{
   /* if no errors, close the input file */
   fclose(input);
}

/* must be 50 to 100 requests in file to work */
if(l_Count >= 50 && l_Count <= 100)
{
   /* create the 4 threads */
   pthread_create(&r_id, NULL, request, (void *)(&l_Count));
   pthread_create(&l1_id, NULL, lift, (void *)(&l1));
   pthread_create(&l2_id, NULL, lift, (void *)(&l2));
   pthread_create(&l3_id, NULL, lift, (void *)(&l3));

   /* exit the threads when they're done */
```

```
        pthread_join(r_id, NULL);
        pthread_join(l1_id, NULL);
        pthread_join(l2_id, NULL);
        pthread_join(l3_id, NULL);

        fprintf(output, "Total number of requests: %d\n", totReq);
        fprintf(output, "Total number of movements: %d\n", totalMove);
    }
    else
    {
        printf("Error! Input should be between 50 and 100 lines\n");
    }

    /* destory mutex and conds when done */
    pthread_mutex_destroy(&lock);
    pthread_mutex_destroy(&fileLock);
    pthread_cond_destroy(&full);
    pthread_cond_destroy(&empty);

    /* free queue and close output file when done*/
    freeQueue(buffer);
    fclose(output);
}

/*
 * SUBMODULE: request
 * IMPORT: lCount(void *)
 * EXPORT: void *
 * ASSERTION: Runs the lift request function, adding to buffer
 * REFERENCE:
 */
void *request(void* lCount)
```

```
{
   int start, dest, numReq;
   Entry* currEnt;
   FILE* input = fopen("sim_input", "r");
   numReq = 1;

   /* loop through entire file */
   while(!feof(input))
   {
      /* get next request and store it */
      fscanf(input, "%d %d", &start, &dest);
      currEnt = (Entry*)malloc(sizeof(Entry));
      currEnt->start = start;
      currEnt->dest = dest;

      /* lock to ensure no other thread is using buffer */
      pthread_mutex_lock(&lock);
      /* if buffer is full, wait */
      if(isFull(buffer))
      {
         pthread_cond_wait(&full, &lock);
      }

      /* enqueue next request onto buffer */
      enqueue(buffer, *currEnt);
      free(currEnt);
      printf("  Request: %d %d\n", start, dest);

      /* free lock and signal to show buffer is no longer empty*/
      pthread_cond_signal(&empty);
      pthread_mutex_unlock(&lock);
```

```
    /* lock the output file so no other thread is using it */
    pthread_mutex_lock(&fileLock);

    /* don't use this output if debugging is defined */
    #ifndef DEBUG_L
    #ifndef DEBUG_R
    fprintf(output, "------------------------------------------\n");
    fprintf(output, "  New Lift Request From Floor %d to Floor %d\n", start, dest);
    fprintf(output, "  Request No: %d\n", numReq);
    fprintf(output, "------------------------------------------\n");
    #endif
    #endif

    /* if debugging request function, print this instead*/
    #ifdef DEBUG_R
    fprintf(output, "%d %d\n", start, dest);
    #endif
    /* unlock file lock */
    pthread_mutex_unlock(&fileLock);

    numReq++;
}

/* if buffer has finished, set done to true */
pthread_mutex_lock(&lock);
setDone(buffer);
printf("!!Done\n");
pthread_mutex_unlock(&lock);

/* close file and exit thread */
fclose(input);
printf("Request has exited\n");
```

```
    totReq = numReq -1;
    pthread_exit(0);
}

/*
 * SUBMODULE: lift
 * IMPORT: tid(void *)
 * EXPORT: void *
 * ASSERTION: Runs the lift functions, removing from the buffer
 * REFERENCE:
 */
void *lift(void* tid)
{
    int start, dest, prevF, totMove, req, fin;
    Entry* currEnt;
    int done = 0;
    int id = *((int *)tid); /* set id for thread to passed int*/
    totMove = 0;
    req = 0;
    currEnt = NULL;

    /* Continue doing requests until finished */
    while(!done)
    {
        fin = 0;

        /* lock so no other thread is using buffer */
        pthread_mutex_lock(&lock);
        /* if buffer is empty and buffer hasn't finished adding requests, wait */
        if(isEmpty(buffer) && !isDone(buffer))
        {
            pthread_cond_wait(&empty, &lock);
```

```
}

/* if buffer is empty don't dequeue */
if(!isEmpty(buffer))
{
   currEnt = dequeue(buffer);
   printf("  Lift-%d: %d %d\n", id, currEnt->start, currEnt->dest);
   start = currEnt->start;
   dest = currEnt->dest;
   free(currEnt);
   fin = 1;
}

/* let requested know its not full anymore */
pthread_cond_signal(&full);
pthread_mutex_unlock(&lock);

/* This is so if singalled and buffer is empty it doesn't print nothing */
if(fin)
{
   /* lock so no other thread is outputting */
   pthread_mutex_lock(&fileLock);
   /* if debug is defined don't do this output */
   #ifndef DEBUG_L
   #ifndef DEBUG_R
   fprintf(output, "Lift-%d Operation\n", id);
   fprintf(output, "Previous position: Floor %d\n", prevF);
   fprintf(output, "Request: Floor %d to Floor %d\n", start, dest);
   fprintf(output, "Detail operations:\n");
   fprintf(output, "   Go from Floor %d to Floor %d\n", prevF, start);
   fprintf(output, "   Go from Floor %d to Floor %d\n", start, dest);
   fprintf(output, "   #movement for this request: %d\n", abs(prevF - start) + abs(dest - start));
```

```
        fprintf(output, "   #request: %d\n", req);
        fprintf(output, "   Total #movement: %d\n", totMove + abs(prevF - start) + abs(dest - start));
        fprintf(output, "Current Position: Floor %d\n", dest);
        #endif
        #endif

        /* if debugging lift function, print this instead */
        #ifdef DEBUG_L
        fprintf(output, "%d %d\n", start, dest);
        #endif
        /* unlock lock on file */
        pthread_mutex_unlock(&fileLock);

        totMove += abs(prevF - start) + abs(dest - start);
        prevF = dest;
        req++;
    }

    /* if buffer is empty and is finished, exit loop */
    pthread_mutex_lock(&lock);
    if(isDone(buffer) && isEmpty(buffer))
    {
        done = 1;
    }
    pthread_mutex_unlock(&lock);



    sleep(sleepT);
}
pthread_cond_signal(&empty);
pthread_cond_signal(&empty);
```

```
    pthread_mutex_lock(&lock);
    totalMove += totMove;
    pthread_mutex_unlock(&lock);

    /* end thread */
    printf("Lift-%d: has exited\n", id);
    pthread_exit(0);
}
```

PART B

circularQueue.h

```
/*
 * File: CircularQueue.h
 * File Created: Tuesday, 5th May 2020
 * Author: Nicholas Klvana-Hooper
 * -----
 * Last Modified: Tuesday, 12th May 2020
 * Modified By: Nicholas Klvana-Hooper
 * -----
 * Purpose: Contains header information for CircularQueue.h
 * Reference:
 */
#ifndef CIRCULARQUEUE_H
#define CIRCULARQUEUE_H

/* Entry struct definition */
typedef struct{
   int start;
   int dest;
} Entry;

/* CircularQueue struct definition */
typedef struct{
   int max;
   int head;
   int tail;
   int count;
   int done;
   Entry* data;
} CircularQueue;
```

```
/* Didn't work so I have prototyped this function */
int ftruncate(int fd, off_t length);

void createCircularQueue(int size);
int isEmpty();
int isFull(CircularQueue* queue);
void enqueue(int source, int dest);
Entry* dequeue();
Entry* peek();
void freeQueue();
void printQueue();
int isDone();
void setDone();

#endif
```

circularQueue.c

```
/*
 * File: circularQueue.c
 * File Created: Thursday, 7th May 2020
 * Author: Nicholas Klvana-Hooper
 * -----
 * Last Modified: Tuesday, 12th May 2020
 * Modified By: Nicholas Klvana-Hooper
 * -----
 * Purpose: Contains methods for a circular queue implementation
 * Reference:
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
#include <string.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "circularQueue.h"

/*
 * SUBMODULE: createCircularQueue
 * IMPORT: size(int)
 * EXPORT: void
 * ASSERTION: Creates the shared memory for and initialises the queue
 * REFERENCE:
 */
void createCircularQueue(int size)
{
   int buff_fd, data_fd, ent_fd, i;
   CircularQueue* queue;
   Entry* temp;
   char num[3];

   /* Open buffer shared memory and map it */
   buff_fd = shm_open("/BUFFER", O_CREAT | O_RDWR, 0666);
   ftruncate(buff_fd, sizeof(CircularQueue));
   queue = (CircularQueue*) mmap(0, sizeof(CircularQueue), PROT_READ | PROT_WRITE,
MAP_SHARED, buff_fd, 0);

   /* Initialise the queue */
   queue->max = size;
   queue->head = 0;
   queue->tail = 0;
   queue->count = 0;
   queue->done = 0;

   /* Open data shared memory and map it */
```

```
   data_fd = shm_open("/DATA", O_CREAT | O_RDWR, 0666);
   ftruncate(data_fd, size * sizeof(Entry));
   queue->data = (Entry*) mmap(0, size * sizeof(Entry), PROT_READ | PROT_WRITE, MAP_SHARED,
data_fd, 0);

   for(i = 0; i < size; i++)
   {
      /* has name "/<num>" ie. "/0", "/1" etc. */
      sprintf(num, "/%d", i);

      /* Open entry shared memory and map it */
      ent_fd = shm_open(num, O_CREAT | O_RDWR, 0666);
      ftruncate(ent_fd, sizeof(Entry));
      temp = (Entry*) mmap(0, sizeof(Entry), PROT_READ | PROT_WRITE, MAP_SHARED, ent_fd, 0);

      /* initialise the blank entry */
      temp->start = 0;
      temp->dest = 0;

      /* unmap the temp shared memory from this function */
      munmap(temp, sizeof(Entry));
   }

   /* unmap the queue shared memory from this function */
   munmap(queue, sizeof(CircularQueue));
}

/*
 * SUBMODULE: isEmpty
 * IMPORT: void
 * EXPORT: bool(int)
 * ASSERTION: Returns an int representing a boolean saying if the queue is empty
 * REFERENCE:
```

```
 */
int isEmpty()
{
   int buff_fd;
   CircularQueue* queue;
   int bool = 0;

   /* Open buffer shared memory and map it */
   buff_fd = shm_open("/BUFFER", O_RDONLY, 0666);
   queue = (CircularQueue*) mmap(0, sizeof(CircularQueue), PROT_READ, MAP_SHARED, buff_fd, 0);

   /* if queue is empty, count = 0 */
   if(queue->count == 0)
   {
      bool = 1;
   }

   /* unmap the queue shared memory from this function */
   munmap(queue, sizeof(CircularQueue));
   return bool;
}

/*
 * SUBMODULE: isFull
 * IMPORT: queue(CircularQueue*)
 * EXPORT: bool(int)
 * ASSERTION: Returns an int representing a boolean saying if the queue is full
 * REFERENCE:
 */
int isFull(CircularQueue* queue)
{
   int bool = 0;
```

```
   if(queue->count == queue->max)
   {
      bool = 1;
   }
   return bool;
}

/*
 * SUBMODULE: enqueue
 * IMPORT: start(int), dest(int)
 * EXPORT: void
 * ASSERTION: Adds a new entry object into the queue
 * REFERENCE:
 */
void enqueue(int start, int dest)
{
   int entry_fd, buff_fd;
   Entry* ent;
   char num[3];
   CircularQueue* queue;

   /* Open buffer shared memory and map it */
   buff_fd = shm_open("/BUFFER", O_CREAT | O_RDWR, 0666);
   queue = (CircularQueue*) mmap(0, sizeof(CircularQueue),  PROT_READ | PROT_WRITE,
MAP_SHARED, buff_fd, 0);

   if(isFull(queue))
   {
      printf("Error: Queue is full!\n");
   }
   else
   {
```

```
        /* has name "/<num>" ie. "/0", "/1" etc. */
        sprintf(num, "/%d", queue->tail);

        /* Open/create array shared memory and map it */
        entry_fd = shm_open(num, O_CREAT | O_RDWR, 0666);
        ftruncate(entry_fd, sizeof(Entry));
        ent = (Entry*) mmap(0, sizeof(Entry),  PROT_READ | PROT_WRITE, MAP_SHARED, entry_fd, 0);

        /* Set start and dest of new entry */
        ent->start = start;
        ent->dest = dest;

        /* perform circular queue function */
        queue->data[queue->tail] = *ent;
        queue->tail = (queue->tail + 1) % queue->max;
        queue->count += 1;
    }

    /* unmap the queue shared memory from this function */
    munmap(queue, sizeof(CircularQueue));
}

/*
 * SUBMODULE: dequeue
 * IMPORT: void
 * EXPORT: ent(Entry*)
 * ASSERTION: Removes the next entry object from the queue
 * REFERENCE:
 */
Entry* dequeue()
{
    int buff_fd;
```

```
    CircularQueue* queue;
    Entry* ent;
    Entry* blank;

    /* Open buffer shared memory and map it */
    buff_fd = shm_open("/BUFFER", O_CREAT | O_RDWR, 0666);
    ftruncate(buff_fd, sizeof(CircularQueue));
    queue = (CircularQueue*) mmap(0, sizeof(CircularQueue),  PROT_READ | PROT_WRITE,
MAP_SHARED, buff_fd, 0);

    /* create a blank entry to replace the deqeueu-d one */
    blank = (Entry*)malloc(sizeof(Entry));
    blank->start = 0;
    blank->dest = 0;

    /* get next entry from queue, then set head to next index */
    ent = peek();
    queue->data[queue->head] = *blank;
    queue->head = (queue->head + 1) % queue->max;

    /* count is one less */
    queue->count -= 1;

    free(blank);

    /* unmap the queue shared memory from this function */
    munmap(queue, sizeof(CircularQueue));
    return ent;
}

/*
 * SUBMODULE: peek
 * IMPORT: void
```

```
 * EXPORT: ent (Entry *)
 * ASSERTION: Take a copy of the next queue in the queue
 * REFERENCE:
 */
Entry* peek()
{
   int buff_fd;
   Entry* ent;
   CircularQueue* queue;

   /* Open buffer shared memory and map it */
   buff_fd = shm_open("/BUFFER", O_RDONLY, 0666);
   queue = (CircularQueue*) mmap(0, sizeof(CircularQueue), PROT_READ, MAP_SHARED, buff_fd, 0);

   ent = (Entry*)malloc(sizeof(Entry));

   if(isEmpty() == 1)
   {
      printf("Error: Queue is empty!\n");
   }
   else
   {
      /* get next entry from the queue */
      *ent = queue->data[queue->head];
   }

   /* unmap the queue shared memory from this function */
   munmap(queue, sizeof(CircularQueue));
   return ent;
}

/*
```

```
 * SUBMODULE: freeQueue
 * IMPORT: void
 * EXPORT: void
 * ASSERTION: Removes all of the shared memory of the queue
 * REFERENCE:
 */
void freeQueue()
{
   int buff_fd, data_fd, i;
   CircularQueue* queue;
   char num[3];

   /* Open buffer shared memory and map it */
   buff_fd = shm_open("/BUFFER", O_RDONLY, 0666);
   queue = (CircularQueue*) mmap(0, sizeof(CircularQueue), PROT_READ, MAP_SHARED, buff_fd, 0);

   for (i = 0; i < queue->max; i++)
   {
      /* has name "/<num>" ie. "/0", "/1" etc. */
      sprintf(num, "/%d", i);

      /* unmap each entry shared memory from this function as well as closing and */
      /* removing it completely */
      data_fd = shm_open(num, O_RDONLY, 0666);
      close(data_fd);
      shm_unlink(num);
   }

   /* unmap the data shared memory from this function as well as closing and */
   /* removing it completely */
   data_fd = shm_open("/DATA", O_RDONLY, 0666);
   close(data_fd);
```

```
    shm_unlink("/DATA");

    /* unmap the queue shared memory from this function as well as closing and */
    /* removing it completely */
    munmap(queue, sizeof(CircularQueue));
    close(buff_fd);
    shm_unlink("/BUFFER");
}

/*
 * SUBMODULE: isDone
 * IMPORT: void
 * EXPORT: void
 * ASSERTION: Returns an int representing a boolean of whether the queue is done inputting
 * REFERENCE:
 */
int isDone()
{
    int buff_fd, bool;
    CircularQueue* queue;

    /* Open buffer shared memory and map it */
    buff_fd = shm_open("/BUFFER", O_RDONLY, 0666);
    queue = (CircularQueue*) mmap(0, sizeof(CircularQueue), PROT_READ, MAP_SHARED, buff_fd, 0);

    /* gets value of done */
    bool = queue->done;

    /* unmap the queue shared memory from this function */
    munmap(queue, sizeof(CircularQueue));
    return bool;
}
```

```
/*
 * SUBMODULE: setDone
 * IMPORT: void
 * EXPORT: void
 * ASSERTION: Sets the done int within the queue
 * REFERENCE:
 */
void setDone()
{
   int buff_fd;
   CircularQueue * queue;

   /* Open buffer shared memory and map it */
   buff_fd = shm_open("/BUFFER", O_CREAT | O_RDWR, 0666);
   ftruncate(buff_fd, sizeof(CircularQueue));
   queue = (CircularQueue*) mmap(0, sizeof(CircularQueue),  PROT_READ | PROT_WRITE,
MAP_SHARED, buff_fd, 0);

   /* sets done within the queue */
   queue->done = 1;

   /* unmap the queue shared memory from this function */
   munmap(queue, sizeof(CircularQueue));
}
```

Lift_sim_b.h

```
/*
 * File: lift_sim_B.h
 * File Created: Thursday, 7th May 2020
 * Author: Nicholas Klvana-Hooper
 * -----
 * Last Modified: Tuesday, 12th May 2020
```

```
 * Modified By: Nicholas Klvana-Hooper
 * -----
 * Purpose: Contains header information for lift_sim_B.c
 * Reference:
 */
#ifndef LIFTSIMB_H
#define LIFTSIMB_H

void simFunc(int sleepT, int buffSize);
void request(void);
void lift(int sleepT);

#endif
```

Lift_sim_b.c

```
/*
 * File: lift_sim_B.c
 * File Created: Thursday, 7th May 2020
 * Author: Nicholas Klvana-Hooper
 * -----
 * Last Modified: Tuesday, 12th May 2020
 * Modified By: Nicholas Klvana-Hooper
 * -----
 * Purpose: Runs a lift 3 lift simulator for a 20 story building
 * Reference:
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>
```

```
#include <string.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "circularQueue.h"
#include "lift_sim_B.h"

int main(int argc, char* argv[])
{
   pid_t l_Count, sleepT;
   char currChar = 'a';
   FILE* input = fopen("sim_input", "r"); /* Open input file to check number of lines */

   if(argc != 3)
   {
      printf("Error! Usage should be ./lift_sim_B <bufferSize> <timeCount\n");
   }
   else
   {
      createCircularQueue(atoi(argv[1])); /* Initialise queue with size of input */
      sleepT = atoi(argv[2]); /* Sleep for requested amount of time */

      if(input == NULL) /* If doesn't exist */
      {
         perror("Error: could not open sim_input\n");
      }
      else if(ferror(input)) /* If error, print */
      {
         perror("Error reading from sim_input\n");
      }
      else
      {
         l_Count = 1; /* Line count starts at one*/
```

```
        while(currChar != EOF) /* Count number of lines in a file */
        {
            currChar = fgetc(input);
            if(currChar == '\n')
            {
                l_Count++;
            }
        }
    }

    if(ferror(input)) /* If Error, print */
    {
        perror("Error when reading from sim_input\n");
    }
    else /* Or close the file */
    {
        fclose(input);
    }

    /* Must be between 50 and 100 requests */
    if(l_Count >= 50 && l_Count <= 100)
    {
        simFunc(sleepT, atoi(argv[1]));
    }
    else
    {
        printf("Error! Input should be between 50 and 100 lines\n");
    }
  }

  return 0;
}
```

```
/*
 * SUBMODULE: simFunc
 * IMPORT: sleepT(int), buffSize(int)
 * EXPORT: void
 * ASSERTION: Runs the forking for the 4 child processes, creating semaphores
 * REFERENCE:
 */
void simFunc(int sleepT, int buffSize)
{
   pid_t rid, l1, l2, l3;
   int full_fd, lock_fd, empty_fd, fileL_fd, totalMove_fd, totalReq_fd;
   int *totalMove, *totalReq;
   sem_t *full, *empty, *lock, *fileL;
   FILE* output;

   /* Creates shared memory for and initialises sempahores */
   full_fd = shm_open("/full", O_CREAT | O_RDWR, 0666);
   ftruncate(full_fd, sizeof(sem_t));
   full = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, full_fd, 0);
   sem_init(full, 1, 0);

   lock_fd = shm_open("/lock", O_CREAT | O_RDWR, 0666);
   ftruncate(lock_fd, sizeof(sem_t));
   lock = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, lock_fd, 0);
   sem_init(lock, 1, 1);

   empty_fd = shm_open("/empty", O_CREAT | O_RDWR, 0666);
   ftruncate(empty_fd, sizeof(sem_t));
   empty = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, empty_fd,
0);
   sem_init(empty, 1, buffSize);
```

```
   fileL_fd = shm_open("/fileL", O_CREAT | O_RDWR, 0666);
   ftruncate(fileL_fd, sizeof(sem_t));
   fileL = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, fileL_fd, 0);
   sem_init(fileL, 1, 1);

   totalReq_fd = shm_open("/totalReq", O_CREAT | O_RDWR, 0666);
   ftruncate(totalReq_fd, sizeof(int));
   totalReq = (int*) mmap(0, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, totalReq_fd, 0);
   *totalReq = 0;

   totalMove_fd = shm_open("/totalMove", O_CREAT | O_RDWR, 0666);
   ftruncate(totalMove_fd, sizeof(int));
   totalMove = (int*) mmap(0, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, totalMove_fd,
0);
   *totalMove = 0;

   /* Forks to create the request child */
   rid = fork();
   if(!rid)
   {
      /* If this is request process, print and run the function */
      printf("Request Process Forked-> pid: %d ppid: %d\n", getpid(), getppid());
      request();
   }
   else
   {
      /* If this is the parent process, fork */
      l1 = fork();
      if(!l1)
      {
         /* If this is lift-1 process, print and run the function */
         printf("Lift-1 Process Forked-> pid: %d ppid: %d\n", getpid(), getppid());
         lift(sleepT);
```

```
}
else
{
   /* If this is the parent process, fork */
   l2 = fork();
   if(!l2)
   {
      /* If this is lift-2 process, print and run the function */
      printf("Lift-2 Process Forked-> pid: %d ppid: %d\n", getpid(), getppid());
      lift(sleepT);
   }
   else
   {
      /* If this is the parent process, fork */
      l3 = fork();
      if(!l3)
      {
         /* If this is lift-3 process, print and run the function */
         printf("Lift-3 Process Forked-> pid: %d ppid: %d\n", getpid(), getppid());
         lift(sleepT);
      }
      else
      {
         /* If this is the parent process, destory semaphores and print */
         printf("I'm the parent process-> pid: %d\n", getpid());
         /* wait for the child processes to finish */
         wait(NULL);
         wait(NULL);
         wait(NULL);
         wait(NULL);

         freeQueue(); /* free all shared memory in queue */
```

```
                output = fopen("sim_output", "a");
                fprintf(output, "Total number of requests: %d\n", *totalReq);
                fprintf(output, "Total number of movements: %d\n", *totalMove);

                shm_unlink("/full_sem");
                shm_unlink("/empty_sem");
                shm_unlink("/lock_sem");
                shm_unlink("/fileL_sem");
                shm_unlink("/totalMove");
                shm_unlink("/totalReq");
            }
        }
    }
  }
}

/*
 * SUBMODULE: request
 * IMPORT: void
 * EXPORT: void
 * ASSERTION: Runs the lift request function, adding to buffer
 * REFERENCE:
 */
void request(void)
{
   int source, dest, full_fd, lock_fd, empty_fd, fileL_fd, totalReq_fd;
   sem_t *full, *lock, *empty, *fileL;
   FILE* output;
   FILE* input = fopen("sim_input", "r");
   int numReq = 1;
   int *totalReq;
```

```
   /* Opens memory to write/read semaphores */
   full_fd = shm_open("/full", O_RDWR, 0666);
   ftruncate(full_fd, sizeof(sem_t));
   full = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, full_fd, 0);

   lock_fd = shm_open("/lock", O_RDWR, 0666);
   ftruncate(lock_fd, sizeof(sem_t));
   lock = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, lock_fd, 0);

   empty_fd = shm_open("/empty", O_RDWR, 0666);
   ftruncate(empty_fd, sizeof(sem_t));
   empty = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, empty_fd,
0);

   fileL_fd = shm_open("/fileL", O_RDWR, 0666);
   ftruncate(fileL_fd, sizeof(sem_t));
   fileL = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, fileL_fd, 0);

   totalReq_fd = shm_open("/totalReq", O_CREAT | O_RDWR, 0666);
   ftruncate(totalReq_fd, sizeof(int));
   totalReq = (int*) mmap(0, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, totalReq_fd, 0);

   /* Continuing loop through entire file */
   while(!feof(input))
   {
      /* Read next line */
      fscanf(input, "%d %d\n", &source, &dest);

      /* Wait/signal on sempahore/lock and write to buffer */
      sem_wait(empty);
      sem_wait(lock);
```

```
    enqueue(source, dest);
    printf("Enqueued: %d, %d\n", source, dest);

    sem_post(lock);
    sem_post(full);

    /* Wait/signal on semaphore/lock to write to output file */
    sem_wait(fileL);

    output = fopen("sim_output", "a"); /* open file to output to */
    #ifndef DEBUG_R /* for debugging request function */
    #ifndef DEBUG_L /* for debugging lift function */
    /* Print to file */
    fprintf(output, "-----------------------------------------\n");
    fprintf(output, "  New Lift Request From Floor %d to Floor %d\n", source, dest);
    fprintf(output, "  Request No: %d\n", numReq);
    fprintf(output, "-----------------------------------------\n");
    #endif
    #endif
    #ifdef DEBUG_R /* for debugging request function */
    fprintf(output, "%d %d\n", source, dest);
    #endif

    /* close file and post semaphore to signifiy finished outputting */
    fclose(output);
    sem_post(fileL);

    numReq++;
}

/* Once requesting has gone through input, set done to 1 */
sem_wait(lock);
```

```
    setDone();
    *totalReq = numReq -1; /* Export totalReq to parent process */
    sem_post(lock);

    /* Unmap semaphores from current function */
    munmap(full, sizeof(sem_t));
    munmap(empty, sizeof(sem_t));
    munmap(lock, sizeof(sem_t));
    munmap(fileL, sizeof(sem_t));
    munmap(totalReq, sizeof(int));

    /* Close semaphores */
    close(full_fd);
    close(lock_fd);
    close(empty_fd);
    close(fileL_fd);
    close(totalReq_fd);

    /* Close input file */
    fclose(input);
}

/*
 * SUBMODULE: lift
 * IMPORT: sleepT(int)
 * EXPORT: void
 * ASSERTION: Runs the lift functions, removing from the buffer
 * REFERENCE:
 */
void lift(int sleepT)
{
    int full_fd, lock_fd, empty_fd, fileL_fd, buff_fd, totalMove_fd, writeNow, start, dest;
```

```
   sem_t *full, *lock, *empty, *fileL;
   Entry *ent;
   FILE* output;
   CircularQueue* buffer;
   int done = 0;
   int* totalMove;

   /* if Debugging don't include this */
   #ifndef DEBUG_R
   #ifndef DEBUG_L
   int prevF = 0;
   int req = 0;
   int totMove = 0;
   #endif
   #endif

   /* Open and map all shared memory for sempahores and buffer */
   full_fd = shm_open("/full", O_CREAT | O_RDWR, 0666);
   ftruncate(full_fd, sizeof(sem_t));
   full = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, full_fd, 0);

   lock_fd = shm_open("/lock", O_CREAT | O_RDWR, 0666);
   ftruncate(lock_fd, sizeof(sem_t));
   lock = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, lock_fd, 0);

   empty_fd = shm_open("/empty", O_CREAT | O_RDWR, 0666);
   ftruncate(empty_fd, sizeof(sem_t));
   empty = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, empty_fd,
0);

   fileL_fd = shm_open("/fileL", O_CREAT | O_RDWR, 0666);
   ftruncate(fileL_fd, sizeof(sem_t));
   fileL = (sem_t*) mmap(0, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED, fileL_fd, 0);
```

```
   buff_fd = shm_open("/BUFFER", O_RDONLY, 0666);
   buffer = (CircularQueue*) mmap(0, sizeof(CircularQueue), PROT_READ, MAP_SHARED, buff_fd, 0);

   totalMove_fd = shm_open("/totalMove", O_CREAT | O_RDWR, 0666);
   ftruncate(totalMove_fd, sizeof(int));
   totalMove = (int*) mmap(0, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, totalMove_fd,
0);

   while(!done)
   {
      start = 0;
      dest = 0;
      writeNow = 0;
      prevF = 0;

      /* If buffer is empty and is not yet finished adding, wait */
      sem_wait(lock);
      if(isEmpty() && !isDone())
      {
         sem_post(lock);
         sem_wait(full);
      }
      else /* Otherwise let go of lock */
      {
         sem_post(lock);
      }

      /* lock and then deque from buffer */
      sem_wait(lock);
      if(!isEmpty())
      {
         ent = dequeue();
```

```
   start = ent->start;
   dest = ent->dest;
   printf("  Lift-%d: %d %d\n", getpid() - getppid() -1, start, dest);
   free(ent);
   writeNow = 1; /* Tells program that this iteration it should print */
            /* So once no more requests, doesn't print anything */
}
sem_post(lock);

/* If an request is dequeued tell request function its not full anymore */
if(writeNow)
{
   sem_post(empty);
}

/* Write to file if dequeued item */
if(writeNow)
{
   sem_wait(fileL); /*Lock file */
   output = fopen("sim_output", "a");

   #ifndef DEBUG_R /* unless debugging, then don't print this */
   #ifndef DEBUG_L
   fprintf(output, "Lift-%d Operation\n", getpid()-getppid()-1);
   fprintf(output, "Previous position: Floor %d\n", prevF);
   fprintf(output, "Request: Floor %d to Floor %d\n", start, dest);
   fprintf(output, "Detail operations:\n");
   fprintf(output, "   Go from Floor %d to Floor %d\n", prevF, start);
   fprintf(output, "   Go from Floor %d to Floor %d\n", start, dest);
   fprintf(output, "   #movement for this request: %d\n", abs(prevF - start) + abs(dest - start));
   fprintf(output, "   #request: %d\n", req);
   fprintf(output, "   Total #movement: %d\n", totMove + abs(prevF - start) + abs(dest - start));
```

```
      fprintf(output, "Current Position: Floor %d\n", dest);
      #endif
      #endif
      #ifdef DEBUG_L /* If debugging lift function, only print this */
      fprintf(output, "%d %d\n", source, dest);
      #endif

      /* Close file and unlock the file lock */
      fclose(output);
      sem_post(fileL);
   }

   /* If buffer is empty and has finished adding, exit loop */
   sem_wait(lock);
   if(isDone() && isEmpty())
   {
      done = 1;
      printf("lift-%d finished\n", getpid() - getppid() -1);
   }
   sem_post(lock);

   #ifndef DEBUG_R /* if debugging don't do this */
   #ifndef DEBUG_L
   totMove += abs(prevF - start) + abs(dest - start);
   prevF = dest;
   req++;
   #endif
   #endif

   sleep(sleepT); /* Sleep for specified time simulating lift movement*/
}
/* When finished tell other two lifts to exit */
```

```
    sem_post(full);
    sem_post(full);

    /* Total move export to parent process */
    sem_wait(lock);
    *totalMove += totMove;
    sem_post(lock);

    /* unmap seamphore and buffer from this function */
    munmap(full, sizeof(sem_t));
    munmap(empty, sizeof(sem_t));
    munmap(lock, sizeof(sem_t));
    munmap(fileL, sizeof(sem_t));
    munmap(buffer, sizeof(CircularQueue));
    munmap(totalMove, sizeof(int));

    /* Close sempahores and buffer */
    close(full_fd);
    close(lock_fd);
    close(empty_fd);
    close(fileL_fd);
    close(buff_fd);
    close(totalMove_fd);
}
```

test.sh

```
#! /bin/bash

# This file checks that program works as expected, file length should be 700 for the 50 line example

make
```

```
valgrind -s ./lift_sim_B 1 0 #simulates with 1 sized buffer and 0 time

wc -l sim_output #prints line lenth

rm sim_output
```

Read Me

To compile part a and b run ‘make’ in both directories.

Part A

Nicholas Klvana-Hooper OSAssignment_PartA Readme

Date created: 7th May 2020

Date last modified: 7th May 2020

Purpose: Simulating a 3 lift setup with 20 floors

Files in project: lift_sim_A.c, lift_sim_A.h, circularQueue.c, circularQueue.h

Test Files: sim_input

Also, run make with DEBUG_L=1 or DEBUG_R=1 R for request thread and L for lift threads

This will give an output same as sim_input but with a new line at the end!

Note: lifts export will be slightly different to the input file. Requests should be exactly same

Functionality: Program take a file (sim_input) with between 50 and 100 lines of lift requests, create 4 threads and execute them

with synchronisation, expoting the details into sim_output

run by going ./lift_sim_A <buffer size> <lift moving time>

Additional functionality: Added debug functionality to program/makefile

Part B

Nicholas Klvana-Hooper OSAssignment_PartB Readme

Date created: 12th May 2020

Date last modified: 12th May 2020

Purpose: Simulating a 3 lift setup with 20 floors

Files in project: lift_sim_B.c, lift_sim_B.h, circularQueue.c, circularQueue.h, test.sh

Test Files: sim_input

Also, run make with DEBUG_L=1 or DEBUG_R=1 R for request thread and L for lift threads

This will give an output same as sim_input but with a new line at the end!

Note: lifts export will be slightly different to the input file. Requests should be exactly same

Functionality: Program take a file (sim_input) with between 50 and 100 lines of lift requests, create 4 threads and execute them

with synchronisation, expoting the details into sim_output

Additional functionality: Added debug functionality to program/makefile

Added a script test.sh file which will test it worked. for a 50 page input should say 700 lines!

run by going ./lift_sim_A <buffer size> <lift moving time>

Mutual Exclusion for process/thread

For part A, we use 4 threads, one to simulate the thread requester and 3 more to simulate the lifts that take a request and do the movement. As part of this, all four threads are accessing a bounded buffer which is shared as a global variable between them all as well as a file pointer to a file in order for them to all add to the end of the file. The requester writes to the buffer, while the three lift threads read and take out from the buffer. As part of my model I used three pthread mutex's and two pthread cond's.

For the requester before the buffer is accesses the buffer lock is locked allowing the requester to have mutual exclusion of the buffer. If the buffer is read as full it waits on the full cond, if not it adds to the buffer, signalling the empty cond and unlocking itself. After this the requester then needs to write to the output file and as part of this it locks the file mutex, writes to file and then unlocks it after. Once the buffer finishes it once again locks and unlocks in order to set the done variable in the buffer to 1.

For the lift threads in this part, the lift first locks the mutex for the buffer and if the buffer is empty and has not finished all the requests in the sim_input file it will wait. It will then take a request out of the buffer and then signal the full mutex saying that the buffer isn't full anymore and once again unlock the mutex lock. This will then also lock the file lock and print its output to the file and then unlock the file lock mutex once again. After doing this the lift thread locks the mutex lock for the buffer once more to do some while loop checking and then unlocks it. Once a thread has exited its loop and finished it will signal empty twice to tell any other threads that they should be finished too and locks and unlocks to change the final total movement for the program.

For part B, there is a parent process for the program which creates a total of 4 child process, one for the request process and three more for the lifts. As part of this simulation all four processes are accessing a total of four semaphores: one for full buffer, empty buffer, a general lock on the buffer and one for the file lock. There is also total request and total movement which are all created using share memory.

For the requester it waits on empty and then the lock, enqueues its request and signals on lock then the full semaphore. The process will then wait for the file lock, output its details to the output file and then signal the file lock as available. At the end of the requester process, once its finished it will wait for the buffer lock to set done to it and then update the shared memory for the total requests made to go back to the parent process for output.

For the lift processes in this part, it will first wait for the buffer lock, then checking if it is empty and the requester process hasn't finished adding to the buffer, if this is true it will unlock the buffer lock and wait on the full lock. It will then wait again for the buffer lock to take off the lift request and perform its required movement and then signal this lock once more. The process if it has gone through this process will then wait for the file lock and perform its necessary outputs and then signal that lock once done. The process waits and checks the buffer to see if the buffer is empty and has finished requesting and then signals. After a lift process finishes it signals twice to tell all other lift processes to finish.

The basic premise of both parts is that only one thread/process should be accessing the buffer or any of the required shared memory at a time. The mutexes and semaphore logic above describes how I utilised them to achieve this so that all shared memory was handled correctly.

Sample input for both parts

```
1 5
7 2
3 8
4 11
12 19
20 7
18 4
6 12
8 19
15 2
2 5
5 3
10 7
4 18
1 20
17 16
14 15
6 3
19 20
3 4
2 8
7 3
9 3
10 2
5 16
2 7
13 12
4 11
6 9
19 2
17 16
7 5
1 2
15 10
8 19
18 7
9 1
15 2
16 6
19 5
6 13
18 15
12 5
18 3
3 7
18 14
20 1
15 5
19 3
6 14
```

Sample output for part a

```
-------------------------------------------
  New Lift Request From Floor 1 to Floor 5
  Request No: 1
-------------------------------------------
Lift-3 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 5
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 5
   #movement for this request: 5
   #request: 0
   Total #movement: 5
Current Position: Floor 5
-------------------------------------------
  New Lift Request From Floor 7 to Floor 2
  Request No: 2
-------------------------------------------
Lift-3 Operation
Previous position: Floor 5
Request: Floor 7 to Floor 2
Detail operations:
   Go from Floor 5 to Floor 7
   Go from Floor 7 to Floor 2
   #movement for this request: 7
   #request: 1
   Total #movement: 12
Current Position: Floor 2
-------------------------------------------
  New Lift Request From Floor 3 to Floor 8
  Request No: 3
-------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 3 to Floor 8
Detail operations:
   Go from Floor 0 to Floor 3
   Go from Floor 3 to Floor 8
   #movement for this request: 8
   #request: 0
   Total #movement: 8
Current Position: Floor 8
-------------------------------------------
  New Lift Request From Floor 4 to Floor 11
  Request No: 4
-------------------------------------------
Lift-3 Operation
Previous position: Floor 2
Request: Floor 4 to Floor 11
```

Detail operations:
  Go from Floor 2 to Floor 4
  Go from Floor 4 to Floor 11
  #movement for this request: 9
  #request: 2
  Total #movement: 21
Current Position: Floor 11

 New Lift Request From Floor 12 to Floor 19
 Request No: 5

Lift-2 Operation
Previous position: Floor 0
Request: Floor 12 to Floor 19
Detail operations:
  Go from Floor 0 to Floor 12
  Go from Floor 12 to Floor 19
  #movement for this request: 19
  #request: 0
  Total #movement: 19
Current Position: Floor 19

 New Lift Request From Floor 20 to Floor 7
 Request No: 6

Lift-2 Operation
Previous position: Floor 19
Request: Floor 20 to Floor 7
Detail operations:
  Go from Floor 19 to Floor 20
  Go from Floor 20 to Floor 7
  #movement for this request: 14
  #request: 1
  Total #movement: 33
Current Position: Floor 7

 New Lift Request From Floor 18 to Floor 4
 Request No: 7

Lift-3 Operation
Previous position: Floor 11
Request: Floor 18 to Floor 4
Detail operations:
  Go from Floor 11 to Floor 18
  Go from Floor 18 to Floor 4
  #movement for this request: 21
  #request: 3
  Total #movement: 42
Current Position: Floor 4

 New Lift Request From Floor 6 to Floor 12

Request No: 8

Lift-2 Operation
Previous position: Floor 7
Request: Floor 6 to Floor 12
Detail operations:
Go from Floor 7 to Floor 6
Go from Floor 6 to Floor 12
#movement for this request: 7
#request: 2
Total #movement: 40
Current Position: Floor 12

New Lift Request From Floor 8 to Floor 19
Request No: 9

Lift-3 Operation
Previous position: Floor 4
Request: Floor 8 to Floor 19
Detail operations:
Go from Floor 4 to Floor 8
Go from Floor 8 to Floor 19
#movement for this request: 15
#request: 4
Total #movement: 57
Current Position: Floor 19

New Lift Request From Floor 15 to Floor 2
Request No: 10

Lift-1 Operation
Previous position: Floor 8
Request: Floor 15 to Floor 2
Detail operations:
Go from Floor 8 to Floor 15
Go from Floor 15 to Floor 2
#movement for this request: 20
#request: 1
Total #movement: 28
Current Position: Floor 2

New Lift Request From Floor 2 to Floor 5
Request No: 11

Lift-2 Operation
Previous position: Floor 12
Request: Floor 2 to Floor 5
Detail operations:
Go from Floor 12 to Floor 2
Go from Floor 2 to Floor 5
#movement for this request: 13

#request: 3
Total #movement: 53
Current Position: Floor 5

New Lift Request From Floor 5 to Floor 3
Request No: 12

Lift-1 Operation
Previous position: Floor 2
Request: Floor 5 to Floor 3
Detail operations:
Go from Floor 2 to Floor 5
Go from Floor 5 to Floor 3
#movement for this request: 5
#request: 2
Total #movement: 33
Current Position: Floor 3

New Lift Request From Floor 10 to Floor 7
Request No: 13

Lift-3 Operation
Previous position: Floor 19
Request: Floor 10 to Floor 7
Detail operations:
Go from Floor 19 to Floor 10
Go from Floor 10 to Floor 7
#movement for this request: 12
#request: 5
Total #movement: 69
Current Position: Floor 7

New Lift Request From Floor 4 to Floor 18
Request No: 14

Lift-2 Operation
Previous position: Floor 5
Request: Floor 4 to Floor 18
Detail operations:
Go from Floor 5 to Floor 4
Go from Floor 4 to Floor 18
#movement for this request: 15
#request: 4
Total #movement: 68
Current Position: Floor 18

New Lift Request From Floor 1 to Floor 20
Request No: 15

Lift-1 Operation
Previous position: Floor 3

Request: Floor 1 to Floor 20
Detail operations:
  Go from Floor 3 to Floor 1
  Go from Floor 1 to Floor 20
  #movement for this request: 21
  #request: 3
  Total #movement: 54
Current Position: Floor 20

 New Lift Request From Floor 17 to Floor 16
 Request No: 16

Lift-2 Operation
Previous position: Floor 18
Request: Floor 17 to Floor 16
Detail operations:
  Go from Floor 18 to Floor 17
  Go from Floor 17 to Floor 16
  #movement for this request: 2
  #request: 5
  Total #movement: 70
Current Position: Floor 16

 New Lift Request From Floor 14 to Floor 15
 Request No: 17

 New Lift Request From Floor 6 to Floor 3
 Request No: 18

Lift-3 Operation
Previous position: Floor 7
Request: Floor 14 to Floor 15
Detail operations:
  Go from Floor 7 to Floor 14
  Go from Floor 14 to Floor 15
  #movement for this request: 8
  #request: 6
  Total #movement: 77
Current Position: Floor 15
Lift-1 Operation
Previous position: Floor 20
Request: Floor 6 to Floor 3
Detail operations:
  Go from Floor 20 to Floor 6
  Go from Floor 6 to Floor 3
  #movement for this request: 17
  #request: 4
  Total #movement: 71
Current Position: Floor 3

New Lift Request From Floor 19 to Floor 20
Request No: 19

Lift-3 Operation
Previous position: Floor 15
Request: Floor 19 to Floor 20
Detail operations:
Go from Floor 15 to Floor 19
Go from Floor 19 to Floor 20
#movement for this request: 5
#request: 7
Total #movement: 82
Current Position: Floor 20

New Lift Request From Floor 3 to Floor 4
Request No: 20

Lift-2 Operation
Previous position: Floor 16
Request: Floor 3 to Floor 4
Detail operations:
Go from Floor 16 to Floor 3
Go from Floor 3 to Floor 4
#movement for this request: 14
#request: 6
Total #movement: 84
Current Position: Floor 4

New Lift Request From Floor 2 to Floor 8
Request No: 21

Lift-1 Operation
Previous position: Floor 3
Request: Floor 2 to Floor 8
Detail operations:
Go from Floor 3 to Floor 2
Go from Floor 2 to Floor 8
#movement for this request: 7
#request: 5
Total #movement: 78
Current Position: Floor 8

New Lift Request From Floor 7 to Floor 3
Request No: 22

Lift-2 Operation
Previous position: Floor 4
Request: Floor 7 to Floor 3
Detail operations:
Go from Floor 4 to Floor 7
Go from Floor 7 to Floor 3

#movement for this request: 7
#request: 7
Total #movement: 91
Current Position: Floor 3

New Lift Request From Floor 9 to Floor 3
Request No: 23

Lift-1 Operation
Previous position: Floor 8
Request: Floor 9 to Floor 3
Detail operations:
Go from Floor 8 to Floor 9
Go from Floor 9 to Floor 3
#movement for this request: 7
#request: 6
Total #movement: 85
Current Position: Floor 3

New Lift Request From Floor 10 to Floor 2
Request No: 24

Lift-3 Operation
Previous position: Floor 20
Request: Floor 10 to Floor 2
Detail operations:
Go from Floor 20 to Floor 10
Go from Floor 10 to Floor 2
#movement for this request: 18
#request: 8
Total #movement: 100
Current Position: Floor 2

New Lift Request From Floor 5 to Floor 16
Request No: 25

Lift-2 Operation
Previous position: Floor 3
Request: Floor 5 to Floor 16
Detail operations:
Go from Floor 3 to Floor 5
Go from Floor 5 to Floor 16
#movement for this request: 13
#request: 8
Total #movement: 104
Current Position: Floor 16

New Lift Request From Floor 2 to Floor 7
Request No: 26

Lift-3 Operation

Previous position: Floor 2
Request: Floor 2 to Floor 7
Detail operations:
  Go from Floor 2 to Floor 2
  Go from Floor 2 to Floor 7
  #movement for this request: 5
  #request: 9
  Total #movement: 105
Current Position: Floor 7

 New Lift Request From Floor 13 to Floor 12
 Request No: 27

Lift-1 Operation
Previous position: Floor 3
Request: Floor 13 to Floor 12
Detail operations:
  Go from Floor 3 to Floor 13
  Go from Floor 13 to Floor 12
  #movement for this request: 11
  #request: 7
  Total #movement: 96
Current Position: Floor 12

 New Lift Request From Floor 4 to Floor 11
 Request No: 28

Lift-2 Operation
Previous position: Floor 16
Request: Floor 4 to Floor 11
Detail operations:
  Go from Floor 16 to Floor 4
  Go from Floor 4 to Floor 11
  #movement for this request: 19
  #request: 9
  Total #movement: 123
Current Position: Floor 11

 New Lift Request From Floor 6 to Floor 9
 Request No: 29

Lift-3 Operation
Previous position: Floor 7
Request: Floor 6 to Floor 9
Detail operations:
  Go from Floor 7 to Floor 6
  Go from Floor 6 to Floor 9
  #movement for this request: 4
  #request: 10
  Total #movement: 109
Current Position: Floor 9

```
-------------------------------------------
  New Lift Request From Floor 19 to Floor 2
  Request No: 30
-------------------------------------------
Lift-1 Operation
Previous position: Floor 12
Request: Floor 19 to Floor 2
Detail operations:
   Go from Floor 12 to Floor 19
   Go from Floor 19 to Floor 2
   #movement for this request: 24
   #request: 8
   Total #movement: 120
Current Position: Floor 2
-------------------------------------------
  New Lift Request From Floor 17 to Floor 16
  Request No: 31
-------------------------------------------
Lift-2 Operation
Previous position: Floor 11
Request: Floor 17 to Floor 16
Detail operations:
   Go from Floor 11 to Floor 17
   Go from Floor 17 to Floor 16
   #movement for this request: 7
   #request: 10
   Total #movement: 130
Current Position: Floor 16
-------------------------------------------
  New Lift Request From Floor 7 to Floor 5
  Request No: 32
-------------------------------------------
Lift-3 Operation
Previous position: Floor 9
Request: Floor 7 to Floor 5
Detail operations:
   Go from Floor 9 to Floor 7
   Go from Floor 7 to Floor 5
   #movement for this request: 4
   #request: 11
   Total #movement: 113
Current Position: Floor 5
-------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 33
-------------------------------------------
Lift-1 Operation
Previous position: Floor 2
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 2 to Floor 1
```

Go from Floor 1 to Floor 2
#movement for this request: 2
#request: 9
Total #movement: 122
Current Position: Floor 2

New Lift Request From Floor 15 to Floor 10
Request No: 34

Lift-2 Operation
Previous position: Floor 16
Request: Floor 15 to Floor 10
Detail operations:
Go from Floor 16 to Floor 15
Go from Floor 15 to Floor 10
#movement for this request: 6
#request: 11
Total #movement: 136
Current Position: Floor 10

New Lift Request From Floor 8 to Floor 19
Request No: 35

Lift-3 Operation
Previous position: Floor 5
Request: Floor 8 to Floor 19
Detail operations:
Go from Floor 5 to Floor 8
Go from Floor 8 to Floor 19
#movement for this request: 14
#request: 12
Total #movement: 127
Current Position: Floor 19

New Lift Request From Floor 18 to Floor 7
Request No: 36

Lift-1 Operation
Previous position: Floor 2
Request: Floor 18 to Floor 7
Detail operations:
Go from Floor 2 to Floor 18
Go from Floor 18 to Floor 7
#movement for this request: 27
#request: 10
Total #movement: 149
Current Position: Floor 7

New Lift Request From Floor 9 to Floor 1
Request No: 37

Lift-2 Operation
Previous position: Floor 10
Request: Floor 9 to Floor 1
Detail operations:
  Go from Floor 10 to Floor 9
  Go from Floor 9 to Floor 1
  #movement for this request: 9
  #request: 12
  Total #movement: 145
Current Position: Floor 1

 New Lift Request From Floor 15 to Floor 2
 Request No: 38

Lift-3 Operation
Previous position: Floor 19
Request: Floor 15 to Floor 2
Detail operations:
  Go from Floor 19 to Floor 15
  Go from Floor 15 to Floor 2
  #movement for this request: 17
  #request: 13
  Total #movement: 144
Current Position: Floor 2

 New Lift Request From Floor 16 to Floor 6
 Request No: 39

Lift-1 Operation
Previous position: Floor 7
Request: Floor 16 to Floor 6
Detail operations:
  Go from Floor 7 to Floor 16
  Go from Floor 16 to Floor 6
  #movement for this request: 19
  #request: 11
  Total #movement: 168
Current Position: Floor 6

 New Lift Request From Floor 19 to Floor 5
 Request No: 40

Lift-2 Operation
Previous position: Floor 1
Request: Floor 19 to Floor 5
Detail operations:
  Go from Floor 1 to Floor 19
  Go from Floor 19 to Floor 5
  #movement for this request: 32
  #request: 13
  Total #movement: 177

Current Position: Floor 5

New Lift Request From Floor 6 to Floor 13
Request No: 41

Lift-3 Operation
Previous position: Floor 2
Request: Floor 6 to Floor 13
Detail operations:
Go from Floor 2 to Floor 6
Go from Floor 6 to Floor 13
#movement for this request: 11
#request: 14
Total #movement: 155
Current Position: Floor 13

New Lift Request From Floor 18 to Floor 15
Request No: 42

Lift-1 Operation
Previous position: Floor 6
Request: Floor 18 to Floor 15
Detail operations:
Go from Floor 6 to Floor 18
Go from Floor 18 to Floor 15
#movement for this request: 15
#request: 12
Total #movement: 183
Current Position: Floor 15

New Lift Request From Floor 12 to Floor 5
Request No: 43

Lift-2 Operation
Previous position: Floor 5
Request: Floor 12 to Floor 5
Detail operations:
Go from Floor 5 to Floor 12
Go from Floor 12 to Floor 5
#movement for this request: 14
#request: 14
Total #movement: 191
Current Position: Floor 5

New Lift Request From Floor 18 to Floor 3
Request No: 44

Lift-3 Operation
Previous position: Floor 13
Request: Floor 18 to Floor 3
Detail operations:

```
  Go from Floor 13 to Floor 18
  Go from Floor 18 to Floor 3
  #movement for this request: 20
  #request: 15
  Total #movement: 175
Current Position: Floor 3
--------------------------------------------
 New Lift Request From Floor 3 to Floor 7
 Request No: 45
--------------------------------------------
Lift-1 Operation
Previous position: Floor 15
Request: Floor 3 to Floor 7
Detail operations:
  Go from Floor 15 to Floor 3
  Go from Floor 3 to Floor 7
  #movement for this request: 16
  #request: 13
  Total #movement: 199
Current Position: Floor 7
--------------------------------------------
 New Lift Request From Floor 18 to Floor 14
 Request No: 46
--------------------------------------------
Lift-2 Operation
Previous position: Floor 5
Request: Floor 18 to Floor 14
Detail operations:
  Go from Floor 5 to Floor 18
  Go from Floor 18 to Floor 14
  #movement for this request: 17
  #request: 15
  Total #movement: 208
Current Position: Floor 14
--------------------------------------------
 New Lift Request From Floor 20 to Floor 1
 Request No: 47
--------------------------------------------
Lift-3 Operation
Previous position: Floor 3
Request: Floor 20 to Floor 1
Detail operations:
  Go from Floor 3 to Floor 20
  Go from Floor 20 to Floor 1
  #movement for this request: 36
  #request: 16
  Total #movement: 211
Current Position: Floor 1
--------------------------------------------
 New Lift Request From Floor 15 to Floor 5
 Request No: 48
```

Lift-1 Operation
Previous position: Floor 7
Request: Floor 15 to Floor 5
Detail operations:
  Go from Floor 7 to Floor 15
  Go from Floor 15 to Floor 5
  #movement for this request: 18
  #request: 14
  Total #movement: 217
Current Position: Floor 5

 New Lift Request From Floor 19 to Floor 3
 Request No: 49

Lift-2 Operation
Previous position: Floor 14
Request: Floor 19 to Floor 3
Detail operations:
  Go from Floor 14 to Floor 19
  Go from Floor 19 to Floor 3
  #movement for this request: 21
  #request: 16
  Total #movement: 229
Current Position: Floor 3

 New Lift Request From Floor 6 to Floor 14
 Request No: 50

Lift-3 Operation
Previous position: Floor 1
Request: Floor 6 to Floor 14
Detail operations:
  Go from Floor 1 to Floor 6
  Go from Floor 6 to Floor 14
  #movement for this request: 13
  #request: 17
  Total #movement: 224
Current Position: Floor 14
Total number of requests: 50
Total number of movements: 670

Sample output for part b

```
-------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 1
-------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 0
   Total #movement: 2
Current Position: Floor 2
-------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 2
-------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 1
   Total #movement: 4
Current Position: Floor 2
-------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 3
-------------------------------------------
Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 1
   Total #movement: 2
Current Position: Floor 2
-------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 4
-------------------------------------------
Lift-3 Operation
Previous position: Floor 0
```

```
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 0
  Total #movement: 2
Current Position: Floor 2
--------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 5
--------------------------------------------
Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 2
  Total #movement: 4
Current Position: Floor 2
--------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 6
--------------------------------------------
Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 3
  Total #movement: 6
Current Position: Floor 2
--------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 7
--------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 3
  Total #movement: 6
Current Position: Floor 2
--------------------------------------------
```

New Lift Request From Floor 1 to Floor 2
Request No: 8

Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
Go from Floor 0 to Floor 1
Go from Floor 1 to Floor 2
#movement for this request: 2
#request: 4
Total #movement: 8
Current Position: Floor 2

New Lift Request From Floor 1 to Floor 2
Request No: 9

Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
Go from Floor 0 to Floor 1
Go from Floor 1 to Floor 2
#movement for this request: 2
#request: 6
Total #movement: 8
Current Position: Floor 2

New Lift Request From Floor 1 to Floor 2
Request No: 10

Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
Go from Floor 0 to Floor 1
Go from Floor 1 to Floor 2
#movement for this request: 2
#request: 7
Total #movement: 10
Current Position: Floor 2

New Lift Request From Floor 1 to Floor 2
Request No: 11

Lift-3 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
Go from Floor 0 to Floor 1
Go from Floor 1 to Floor 2

```
  #movement for this request: 2
  #request: 5
  Total #movement: 4
Current Position: Floor 2
-------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 12
-------------------------------------------
Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 7
  Total #movement: 10
Current Position: Floor 2
-------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 13
-------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 8
  Total #movement: 12
Current Position: Floor 2
-------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 14
-------------------------------------------
Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 8
  Total #movement: 12
Current Position: Floor 2
-------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 15
-------------------------------------------
Lift-1 Operation
```

Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 9
  Total #movement: 14
Current Position: Floor 2

 New Lift Request From Floor 1 to Floor 2
 Request No: 16

Lift-3 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 7
  Total #movement: 6
Current Position: Floor 2

 New Lift Request From Floor 1 to Floor 2
 Request No: 17

Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 9
  Total #movement: 14
Current Position: Floor 2

 New Lift Request From Floor 1 to Floor 2
 Request No: 18

Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 10
  Total #movement: 16
Current Position: Floor 2

```
-------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 19
-------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 10
   Total #movement: 16
Current Position: Floor 2
-------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 20
-------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 11
   Total #movement: 18
Current Position: Floor 2
-------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 21
-------------------------------------------
Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 12
   Total #movement: 18
Current Position: Floor 2
-------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 22
-------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
```

```
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 12
  Total #movement: 20
Current Position: Floor 2
--------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 23
--------------------------------------------
Lift-3 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 10
  Total #movement: 8
Current Position: Floor 2
--------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 24
--------------------------------------------
Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 13
  Total #movement: 20
Current Position: Floor 2
--------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 25
--------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 13
  Total #movement: 22
Current Position: Floor 2
--------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 26
--------------------------------------------
```

Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 14
  Total #movement: 24
Current Position: Floor 2

 New Lift Request From Floor 1 to Floor 2
 Request No: 27

Lift-3 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 11
  Total #movement: 10
Current Position: Floor 2

 New Lift Request From Floor 1 to Floor 2
 Request No: 28

Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 15
  Total #movement: 26
Current Position: Floor 2

 New Lift Request From Floor 1 to Floor 2
 Request No: 29

Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 16
  Total #movement: 28

```
Current Position: Floor 2
--------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 30
--------------------------------------------
Lift-3 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 13
   Total #movement: 12
Current Position: Floor 2
--------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 31
--------------------------------------------
Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 16
   Total #movement: 22
Current Position: Floor 2
--------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 32
--------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 18
   Total #movement: 30
Current Position: Floor 2
--------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 33
--------------------------------------------
Lift-3 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
```

```
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 14
  Total #movement: 14
Current Position: Floor 2
--------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 34
--------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 19
  Total #movement: 32
Current Position: Floor 2
--------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 35
--------------------------------------------
Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 18
  Total #movement: 24
Current Position: Floor 2
--------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 36
--------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 20
  Total #movement: 34
Current Position: Floor 2
--------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 37
```

```
---------------------------------------------
Lift-3 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 17
  Total #movement: 16
Current Position: Floor 2
---------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 38
---------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 21
  Total #movement: 36
Current Position: Floor 2
---------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 39
---------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 22
  Total #movement: 38
Current Position: Floor 2
---------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 40
---------------------------------------------
Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 22
```

```
  Total #movement: 26
Current Position: Floor 2
-------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 41
-------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 23
  Total #movement: 40
Current Position: Floor 2
-------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 42
-------------------------------------------
Lift-3 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 18
  Total #movement: 18
Current Position: Floor 2
-------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 43
-------------------------------------------
Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 23
  Total #movement: 28
Current Position: Floor 2
-------------------------------------------
 New Lift Request From Floor 1 to Floor 2
 Request No: 44
-------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
```

Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 24
  Total #movement: 42
Current Position: Floor 2

 New Lift Request From Floor 1 to Floor 2
 Request No: 45

Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 25
  Total #movement: 44
Current Position: Floor 2

 New Lift Request From Floor 1 to Floor 2
 Request No: 46

Lift-2 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 24
  Total #movement: 30
Current Position: Floor 2

 New Lift Request From Floor 1 to Floor 2
 Request No: 47

Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
  Go from Floor 0 to Floor 1
  Go from Floor 1 to Floor 2
  #movement for this request: 2
  #request: 26
  Total #movement: 46
Current Position: Floor 2

 New Lift Request From Floor 1 to Floor 2

```
  Request No: 48
---------------------------------------------
Lift-3 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 20
   Total #movement: 20
Current Position: Floor 2
---------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 49
---------------------------------------------
Lift-3 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 21
   Total #movement: 22
Current Position: Floor 2
---------------------------------------------
  New Lift Request From Floor 1 to Floor 2
  Request No: 50
---------------------------------------------
Lift-1 Operation
Previous position: Floor 0
Request: Floor 1 to Floor 2
Detail operations:
   Go from Floor 0 to Floor 1
   Go from Floor 1 to Floor 2
   #movement for this request: 2
   #request: 28
   Total #movement: 48
Current Position: Floor 2
Total number of requests: 50
Total number of movements: 100
```