```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include "integer.h"
#include "queue.h"
#include "stack.h"
#include "comparator.h"
#include "readin.h"
#include "real.h"
#include "string.h"
#include "scanner.h"
#include "people.h"
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>
#include <time.h>
#include <ctype.h>
/*
Raychel Delaney 426-Fall 2017
Used the data structures library created in CS201-Spring 2017
Basically used a queue to store mechLot, oilLot,carsLot
3 different threads running through
each has a condition signal
I had a status inside the person class(class I created for the cars/customers) called "sig"
Honestly I had this all done before the last day and talked to one person realizing I was
completely off
so I redid the whole thing
rip me
*/
int size_val;
int run = 1;
pthread_mutex_t mechLock = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t oilLock = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t cusLock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t mechSig = PTHREAD_COND_INITIALIZER;
pthread_cond_t oilSig =PTHREAD_COND_INITIALIZER;
pthread_cond_t cusSig =PTHREAD_COND_INITIALIZER;
int mechSleep=0;
int oilSleep=0;


typedef void (*Printer)(FILE*,void*);

Printer print;
FILE *fp;
```

```
queue *mechLot;
queue *oilLot;
queue *carsLot;

void *oilRunner() {
   while(1){
      pthread_mutex_lock(&oilLock);
      if(sizeQueue(oilLot)>0){
         people* p = dequeue(oilLot);
         printf("Customer %s - (MCL) is being serviced by the oil tech for %d seconds\n",
p->name, p->oilchange_dur);
         sleep(p->oilchange_dur);
         pthread_mutex_unlock(&oilLock);
         pthread_mutex_lock(&cusLock);
         p->sig=1;
         pthread_mutex_unlock(&cusLock);
         pthread_cond_signal(&cusSig);
      }
      else{
         oilSleep=0;
         pthread_mutex_unlock(&oilLock);

      }
      pthread_mutex_lock(&oilLock);
      while(oilSleep){
         pthread_cond_wait(&oilSig,&oilLock);
      }

      pthread_mutex_unlock(&oilLock);
   }
}

void *mechRunner() {
   while(1){
      pthread_mutex_lock(&mechLock);
      if(sizeQueue(mechLot)>0){
         people* p = dequeue(mechLot);
         printf("Customer %s - (MC) is being serviced by the mechanic for %d
seconds\n", getPeopleName(p), getPeopleMechanic(p));
         sleep(getPeopleMechanic(p));
         pthread_mutex_unlock(&mechLock);
         pthread_mutex_lock(&cusLock);
         p->sig=1;
         pthread_mutex_unlock(&cusLock);
         pthread_cond_signal(&cusSig);
      }
```

```
        else{
            mechSleep=0;
            pthread_mutex_unlock(&mechLock);

        }
        pthread_mutex_lock(&mechLock);
        while(mechSleep){
            pthread_cond_wait(&mechSig,&mechLock);
        }

        pthread_mutex_unlock(&mechLock);
    }
}
void *customerRunner(void *arg){
    pthread_mutex_lock(&mechLock);
    people* p = (people *) arg;
    printf("Customer %s - (MA) arrival\n", p->name);
    if(sizeQueue(mechLot)>=size_val){
        printf("Customer %s - (MZ) leaves busy car maintenance shop\n", p->name);
        pthread_mutex_unlock(&mechLock);
        pthread_exit(0);
    }
    else{
        if(sizeQueue(mechLot)>0){
            printf("Customer %s - (MB1) is waiting for mechanic\n", p->name);
        }
        enqueue(mechLot,p);
        if(mechSleep==0){
            mechSleep=1;
            pthread_mutex_unlock(&mechLock);
            pthread_cond_signal(&mechSig);
        }
        else{
            pthread_mutex_unlock(&mechLock);
        }
        pthread_mutex_lock(&cusLock);
        while(!p->sig){
            pthread_cond_wait(&cusSig,&cusLock);
        }
        pthread_mutex_unlock(&cusLock);
        pthread_mutex_lock(&mechLock);
        if(sizeQueue(mechLot)>0){
            people *q=peekQueue(mechLot);
            if(strcmp(q->name,p->name)!=0){
                printf("Customer %s - (MC1) notifying customer %s that they are next for
mechanic\n", p->name,q->name);
```

```
            printf("Customer %s - (MB2) is no longer waiting for mechanic; signaled by
%s\n", q->name, p->name);
          }
       }
       pthread_mutex_unlock(&mechLock);
       printf("Customer %s - (MD) is done with mechanic\n", p->name);
       if(p->oilchange_dur!=0){
          if(sizeQueue(oilLot)>0){
             printf("Customer %s - (MBL1) is waiting for oil change tech \n", p->name);
          }
       enqueue(oilLot,p);
       if(oilSleep==0){
          oilSleep=1;
          pthread_mutex_unlock(&oilLock);
          pthread_cond_signal(&oilSig);
       }
       else{
          pthread_mutex_unlock(&oilLock);
       }
       pthread_mutex_lock(&cusLock);
       while(!p->sig){
          pthread_cond_wait(&cusSig,&cusLock);
       }
       pthread_mutex_unlock(&cusLock);
       pthread_mutex_lock(&oilLock);
       if(sizeQueue(oilLot)>0){
          people *k=peekQueue(oilLot);
          printf("Customer %s - (MCL1) notifying customer %s that they are next for oil
change tech\n", p->name, k->name);
          printf("Customer %s - (MBL2) is no longer waiting for oil change tech; signaled
by %s\n",k->name,p->name);
       }
       pthread_mutex_unlock(&oilLock);
       printf("Customer %s - (ML) is done with oil change tech\n", p->name);
       }
       printf("Customer %s - (ME) is leaving shop\n", p->name);
   }
   pthread_exit(0);
}

int main() {
       fp = fopen("jobcard", "r");
       print=displayPeople;
       mechLot=newQueue(print);
       oilLot=newQueue(print);
       carsLot=newQueue(print);
```

```
        char *name=(char *)malloc(sizeof(50));
        fscanf(fp,"%d",&size_val);
        while(fgetc(fp)!='\n');
        pthread_t oilThread;
        pthread_t mechThread;
        pthread_t carThread;
        pthread_create(&oilThread, NULL, oilRunner, NULL);
        pthread_create(&mechThread, NULL, mechRunner, NULL);

        int prev=0;
        int i=0;
        int arr,mech,oil;
        while(fscanf(fp,"%[^,],%d,%d,%d\n",name,&arr,&mech,&oil)==4){
           sleep(arr-prev);
           prev=arr;
           pthread_create(&carThread, NULL, customerRunner, newPeople(name, arr,
mech, oil));
           enqueue(carsLot,&carThread);
           name=malloc(sizeof(char)*50);
           i++;
        }
        int j;
        for (j=0; j<i; j++) {
           pthread_join(&carThread[j], NULL);
        }
    return 0;
}


//people.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>
#include <time.h>
#include <ctype.h>
#include "people.h"

people *newPeople(char *x, int arr, int mech, int oil){
    people *p = malloc(sizeof(people));
    if (p == 0){
        fprintf(stderr,"out of memory\n");
        exit(-1);}
```

```
    p->name=malloc(sizeof(strlen(x)));
    strcpy(p->name,x);
    p->arrival=arr;
    p->mechanic_wait=mech;
    p->oilchange_dur=oil;
    p->sig=1;
    return p;
        }
int getPeopleArrival(people *v){
    return v->arrival;
    }
int getPeopleMechanic(people *v){
    return v->mechanic_wait;
        }
int getPeopleOil(people *v){
    return v->oilchange_dur;
        }
int setPeopleArrival(people *v,int x){
    int old = v->arrival;
    v->arrival = x;
    return old;
        }
int setPeopleMechanic(people *v,int x){
    int old = v->mechanic_wait;
    v->mechanic_wait = x;
    return old;
        }
int setPeopleOil(people *v,int x){
    int old = v->oilchange_dur;
    v->oilchange_dur = x;
    return old;
        }

void displayPeopleArrival(FILE *fp,void *v){
    fprintf(fp,"%d",getPeopleArrival((people *) v));
        }
void displayPeopleMechanic(FILE *fp,void *v){
    fprintf(fp,"%d",getPeopleMechanic((people *) v));
        }
void displayPeopleOil(FILE *fp,void *v){
    fprintf(fp,"%d",getPeopleOil((people *) v));
        }

int comparePeopleArrival(void *v,void *w){
    return ((people *) v)->arrival - ((people *) w)->arrival;
    }
```

```
int comparePeopleMechanic(void *v,void *w){
   return ((people *) v)->mechanic_wait - ((people *) w)->mechanic_wait;
   }
int comparePeopleOil(void *v,void *w){
   return ((people *) v)->mechanic_wait - ((people *) w)->mechanic_wait;
   }

char *getPeopleName(people *v){
   return v->name;
   }

char *setPeopleName(people *v,char *x){
   people *old = malloc(sizeof(people));
   old->name=malloc(sizeof(v->name));
   strcpy(old->name,v->name);
   v->name = malloc(sizeof(strlen(x)));
   strcpy(v->name,x);
   return old->name;
         }

void displayPeopleName(FILE *fp,void *v){
   fprintf(fp,"%s",getPeopleName((people *) v));
         }
void displayPeople(FILE *fp,void *v){
   fprintf(fp,"%s, %d, %d, %d",getPeopleName((people *) v),getPeopleArrival((people
*) v),getPeopleMechanic((people *) v),getPeopleOil((people *) v) );
   }

void freePeople(people *v){
   free(v);
         }

//people.h
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>
#include <time.h>
#include <ctype.h>

#ifndef __PEOPLE_INCLUDED__
#define __PEOPLE_INCLUDED__

typedef struct people{
   char *name;
   int arrival;
```

```
    int mechanic_wait;
    int oilchange_dur;
    int sig;
} people;

extern people *newPeople(char *, int, int, int);
extern int getPeopleArrival(people *);
extern int setPeopleArrival(people *,int);
extern int getPeopleMechanic(people *);
extern int setPeopleMechanic(people *,int);
extern int getPeopleOil(people *);
extern int setPeopleOil(people *,int);
extern char *getPeopleName(people *);
extern char *setPeopleName(people *,char *);
extern void displayPeopleName(FILE *,void *);
extern void displayPeopleArrival(FILE *,void *);
extern void displayPeopleMechanic(FILE *,void *);
extern void displayPeopleOil(FILE *,void *);
extern void displayPeople(FILE *,void *);
extern int comparePeopleArrival(void *,void *);
extern int comparePeopleMechanic(void *,void *);
extern int comparePeopleWait(void *,void *);
extern void freePeople(people *);

#define PINFINITY IN_MAX
#define NINFINITY IN_MIN

#endif

//queue.c
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"
#include "sll.h"
#include "integer.h"

queue *newQueue(void (*d)(FILE *,void *)){
        queue *theQueue = malloc(sizeof(queue));
        theQueue->items=newSLL(d);
        return theQueue;
}   //constructor
void enqueue(queue *items,void *value){
        insertSLL(items->items,sizeQueue(items),value);
}     //stores a generic value
void *dequeue(queue *items){
```

```
        void *p= removeSLL(items->items, 0);
        return p;
}           //returns a generic value
void *peekQueue(queue *items){
        void *p= getSLL(items->items,0);
        return p;
}           //returns a generic value
int sizeQueue(queue *items){
        int count= sizeSLL(items->items);
        return count;
}
void displayQueue(FILE *fp,queue *items){
        displaySLL(fp,items->items);
}

//queue.h

#ifndef __QUEUE_INCLUDED__
#define __QUEUE_INCLUDED__
#include "sll.h"

typedef struct queue{
   sll *items;
} queue;

queue *newQueue(void (*d)(FILE *,void *));   //constructor
void enqueue(queue *items,void *value);      //stores a generic value
void *dequeue(queue *items);                 //returns a generic value
void *peekQueue(queue *items);               //returns a generic value
int sizeQueue(queue *items);
void displayQueue(FILE *fp,queue *items);

#endif

//sll.c
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include "sll.h"
#include "integer.h"

sll *newSLL(void (*d)(FILE *,void *)) {
        sll *items = malloc(sizeof(sll));
        if (items == 0){
                fprintf(stderr,"out of memory");
                exit(-1);
```

```
        }
        items->head = 0;
        items->tail = 0;
        items->size = 0;
        items->display = d;
        return items;
}       //d is the display function
void insertSLL(sll *items,int index,void *value){
        if(items->size == 0 && index==0){
                sllnode *node= malloc(sizeof(sllnode));
                items->head=node;
                items->tail=node;
                items->size=1;
                node->next=0;
                node->value=value;
        }
        else if(index > items->size){
                return;
        }
        else if(index==0 && items->size >= 1){
                sllnode *node= malloc(sizeof(sllnode));
                node->value=value;
                node->next=items->head;
                items->head=node;
                items->size= items->size + 1;
        }
        else if(index == items->size){
                sllnode *node= malloc(sizeof(sllnode));
                node->value=value;
                items->tail->next=node;
                items->tail=node;
                node->next=0;
                items->size= items->size + 1;
        }
        else {
                sllnode *node= malloc(sizeof(sllnode));
                node->value=value;
                sllnode *tempN=items->head;
                for(int i=0;i<index-1;i++){
                        tempN=tempN->next;
                }
                node->next=tempN->next;
                tempN->next=node;
                items->size= items->size + 1;
        }
} //stores a generic value
```

```
void *removeSLL(sll *items,int index){
	if(items->size == 0){
		exit(-1);
	}
	else if(index > items->size){
		exit(-1);
	}
	else if(index==0 && items->size >= 1){
		sllnode *node= items->head;
		items->head= items->head->next;
		items->size= items->size -1;
		return node->value;
	}
	else if(index == items->size - 1){
		sllnode *tempN= items->head;
		for(int i=0;i < sizeSLL(items)-2;i++){
			tempN=tempN->next;
		}
		sllnode *node=items->tail;
		tempN->next=0;
		items->tail=tempN;
		items->size= items->size - 1;
		return node->value;
	}
	else{
		sllnode *tempN=items->head;
		for(int i=0;i<index - 1;i++){
			tempN=tempN->next;
		}
		sllnode *node= tempN->next;
		tempN->next=tempN->next->next;
		items->size= items->size - 1;
		return node->value;
	}
}	//returns a generic value
void unionSLL(sll *recipient,sll *donor){
	if(donor->head == NULL){
		return;
	}
	else if(recipient->head==NULL){
		recipient->head=donor->head;
	}
	else{
		recipient->tail->next=donor->head;
	}
	recipient->size=sizeSLL(recipient) + sizeSLL(donor);
```

```
        recipient->tail=donor->tail;
        donor->size=0;
        donor->head=NULL;
        donor->tail=NULL;
}       //merge two lists into one
void *getSLL(sll *items,int index){
        sllnode *node=items->head;
        for (int i = 0; i < index; ++i){
                node=node->next;
        }
        return node->value;
}           //get the value at the index
int sizeSLL(sll *items){
        return items->size;
}
void displaySLL(FILE *f,sll *items){
        sllnode *tempN=items->head;
        fprintf(f,"[");
        if (items->head !=NULL){
                while(tempN!=NULL){
                        items->display(f,tempN->value);
                        if(tempN->next!=NULL){
                                fprintf(f,",");
                        }
                        tempN=tempN->next;
                }
        }
        fprintf(f,"]");
}

//sll.h

#ifndef __SLL_INCLUDED__
#define __SLL_INCLUDED__

typedef struct sllnode{
   void *value;
   struct sllnode *next;
} sllnode;

typedef struct sll{
   sllnode *head;
   sllnode *tail;
   int size;
   void (*display)(FILE *,void *);
} sll;
```

```
extern sll *newSLL(void (*d)(FILE *,void *));           //constructor
extern void insertSLL(sll *items,int index,void *value); //stores a generic value
extern void *removeSLL(sll *items,int index);           //returns a generic value
extern void unionSLL(sll *recipient,sll *donor);        //merge two lists into one
extern void *getSLL(sll *items,int index);              //get the value at the index
extern int sizeSLL(sll *items);
extern void displaySLL(FILE *,sll *items);

#endif
```

//Makefile

```
OBJS = scanner.o sll.o dll.o stack.o queue.o integer.o comparator.o project1a.o readin.o
real.o string.o people.o

OOPTS = -Wall -Wextra -g -c -std=c99
LOPTS = -Wall -Wextra -g -std=c99

all : project1

project1 : $(OBJS)
	gcc $(LOPTS) -o project1 $(OBJS) -lpthread

scanner.o : scanner.c scanner.h
	gcc $(OOPTS) scanner.c

sll.o : sll.c sll.h
	gcc $(OOPTS) sll.c

dll.o : dll.c dll.h
	gcc $(OOPTS) dll.c

stack.o : stack.c stack.h
	gcc $(OOPTS) stack.c

comparator.o : comparator.c comparator.h
	gcc $(OOPTS) comparator.c

queue.o : queue.c queue.h
	gcc $(OOPTS) queue.c

integer.o : integer.c integer.h
	gcc $(OOPTS) integer.c

real.o : real.c real.h
```

```
	gcc $(OOPTS) real.c

people.o : people.c people.h
	gcc $(OOPTS) people.c

string.o : string.c string.h
	gcc $(OOPTS) string.c

readin.o : readin.c readin.h
	gcc $(OOPTS) readin.c

project1a.o : project1a.c integer.h sll.h dll.h stack.h queue.h comparator.h scanner.h
readin.h real.h string.h people.h
	gcc $(OOPTS) -c project1a.c

test : project1

clean  :
	rm -f *.o project1
```