Assignment 1 - The Wacky Store

- Due Feb 25 by 11:59p.m.
- Points 100
- Submitting a file upload
- File Types c
- Available Feb 5 at 12a.m. - Feb 25 at 11:59p.m.

This assignment was locked Feb 25 at 11:59p.m..

Assignment 1 - The Wacky Store

The Wacky Store (TWS) is a vibrant discount wholesale retailer offering a diverse range of products but with a unique twist – no membership fees required! Established in 2006 by Charles, the store quickly became a one-stop shop where customers could find anything and everything, from gaming essentials like Riot RP to digital currencies like Bitcoin.

In 2024, TWS underwent a significant change in ownership, being acquired by the University of Toronto. The students of CSCA48 have been assigned a software engineering position and have been tasked to design a mock store queue simulator.

Get the starter code here: a1.zip (https://q.utoronto.ca/courses/332080/files/30257728?wrap=1) (https://q.utoronto.ca/courses/332080/files/30257728/download?download_frd=1)

Notes:

- Anytime we mention sorting of strings (char[]), we want you to sort using the ASCII table. This means making a call to `strcmp()`.
- You can make your own helper functions. However, **do not change the structs**.
- However, when submitting, do not submit a file with a `main()` function.
  - The main function is purposely kept in a separate file so you can modify it however you wish.
- Additional notes and clarifications will be posted on Piazza. Please ask questions and follow the forums for updates!
- Your file must compile with: `gcc -std=c99 -Wall -Werror -lm main.c -o main`

Understanding the Data:

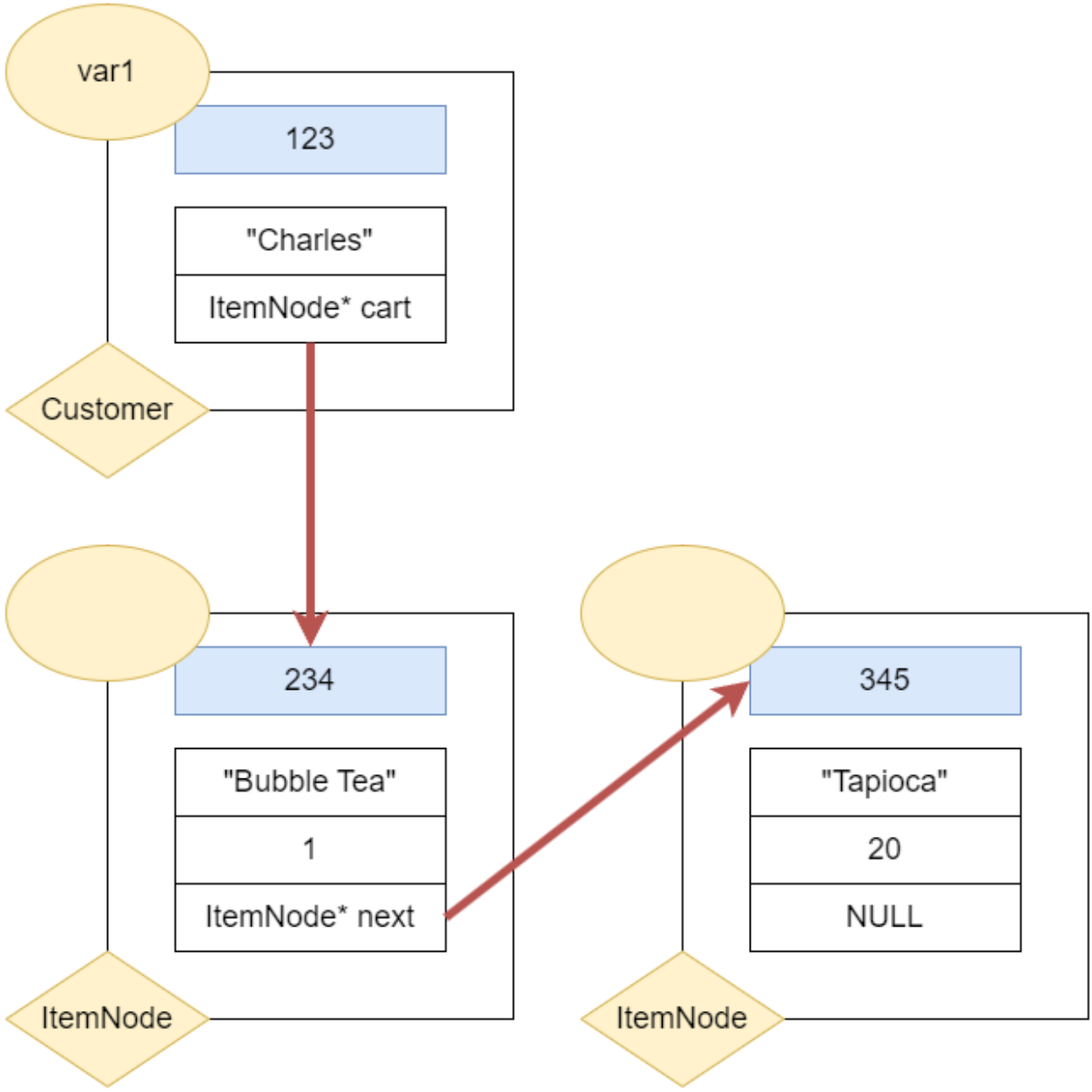
Assignment 1 has a strong focus on using Linked Lists to represent data in the real world.

Customers and Items:

- All **customers** in TWS are identified by a name.
- Each customer entering the store has a cart, which contains a list of items, represented as a linked list.
- All **items** in TWS are identified by **an unique case-sensitive name**.
- At **no point** should any item in the cart have the same name as another in the **same** cart.

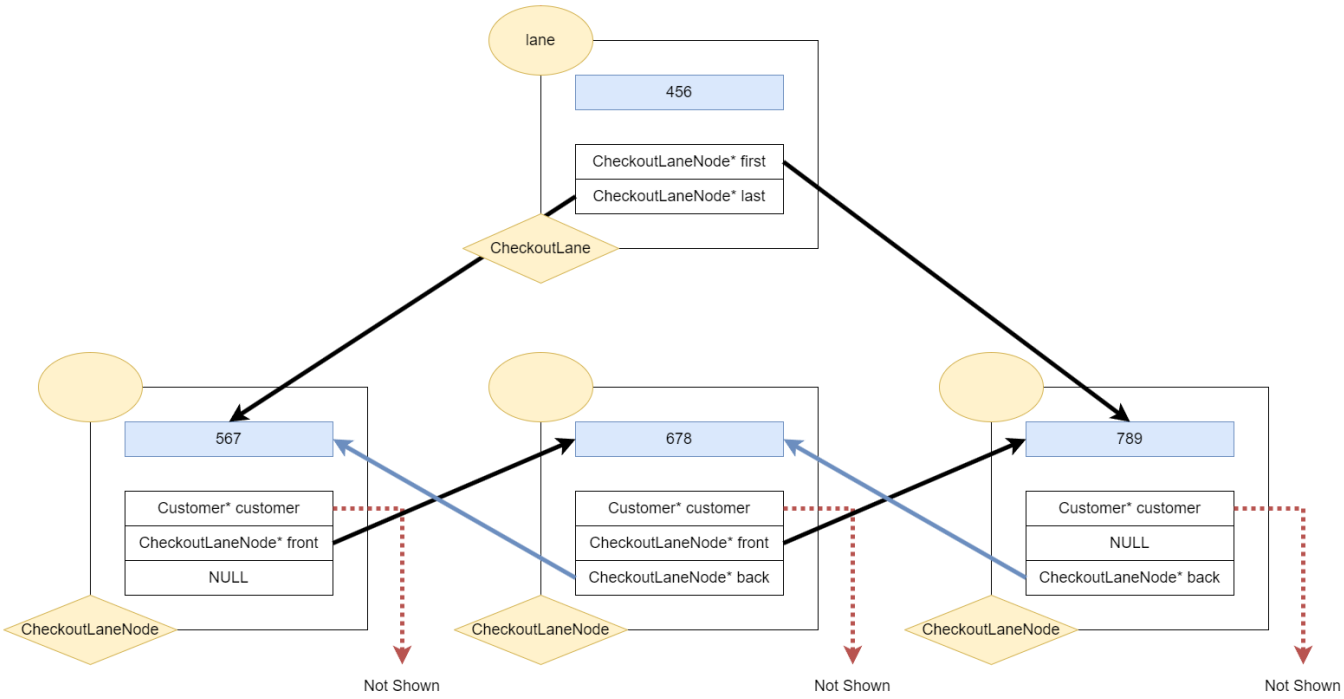
- This means that if Charles has `2 x WATER` and adds `3 x WATER`, then he ends with `5 x WATER`.

- A customer's cart should **always be sorted by ASCII order** (this means you should be using the `strcmp()` function).



Checkout Lanes:

- Checkout lanes are a **Queue FIFO ordered data structure**. Customers are processed in the order they are queued.
- `CheckoutLane` must always keep an accurate reference to the first and last customer in the queue.
- If there are no customers in the lane, the first and last customer reference is `NULL`.



Understanding the Starter Code:

The following lists the functions in the starter code as well as their respective documentation.

```
ItemNode* new_item_node(char* name, int count)
```

Allocate a new ItemNode. Allocation must be done manually (with malloc or calloc). Initialize all variables using the arguments provided. Assume that count will always be greater than 0.

```
Customer* new_customer(char* name)
```

Allocate a new Customer. Allocation must be done manually (with malloc or calloc). Initialize all variables using the arguments provided.

```
void free_customer(Customer* customer)
```

Release all memory associated with a Customer back to the system. This includes any items they may have had in their cart.

```
CheckoutLane* open_new_checkout_line()
```

Allocate a new empty checkout lane. Allocation must be done manually (with malloc or calloc).

```
CheckoutLaneNode* new_checkout_node(Customer* customer)
```

Allocate a new CheckoutLaneNode. Allocation must be done manually (with malloc or calloc). Initialize all variables using the arguments provided. Do not allocate a new customer; instead copy the existing reference over.

```
void add_item_to_cart(Customer* customer, char* item_name, int amount)
```

Add an item to a customer's cart, given its name and amount.

If the given amount is 0 or less, do nothing.

IMPORTANT: The items in a customer's cart should always be arranged in lexicographically smallest order based on the item names. Consider the use of the ASCII `strcmp()` function from `<string.h>`.

No two `ItemNodes` in a customer's cart can have the same name. If the customer already has an `ItemNode` with the same item name in their cart, increase the node's `count` by the given amount instead.

```
void remove_item_from_cart(Customer* customer, char* item_name, int amount)
```

Attempt to reduce the quantity of an item in a customer's cart, given its name and amount.

If no `ItemNode` in the customer's cart match the given item name, or the amount given is <= 0, do nothing.

If the quantity is reduced to a value less than or equal to 0, remove the `ItemNode` from the customer's cart.

- This means you will need to do memory cleanup as well.

```
int total_number_of_items(Customer* customer)
```

Count the total number of items in a customer's cart by summing all `ItemNodes` and their associated quantities.

```
void queue(Customer* customer, CheckoutLane* lane)
```

A customer has finished shopping and wishes to checkout. Add the given customer to the end of the given checkout lane.

```
int process(CheckoutLane* lane)
```

Serve the customer at the head of the checkout lane and return the `total_number_of_items()` the customer had in their cart.

The customer leaves the store after being processed. Free any memory associated with them.

If this function is called on an empty lane, return 0.

```
bool balance_lanes(CheckoutLane* lanes[], int number_of_lanes)
```

Have you've ever been to a store and noticed that some checkout lanes are much busier than the others?

TWS is an extremely busy store, so the manager likes to keep all lanes as balanced as possible.

Write an algorithm to **move a single customer** from **the end** of **the most busy** checkout lane to **the end** of **the least busy** checkout lane.

Busyness is defined as the total number of customers in a checkout lane.

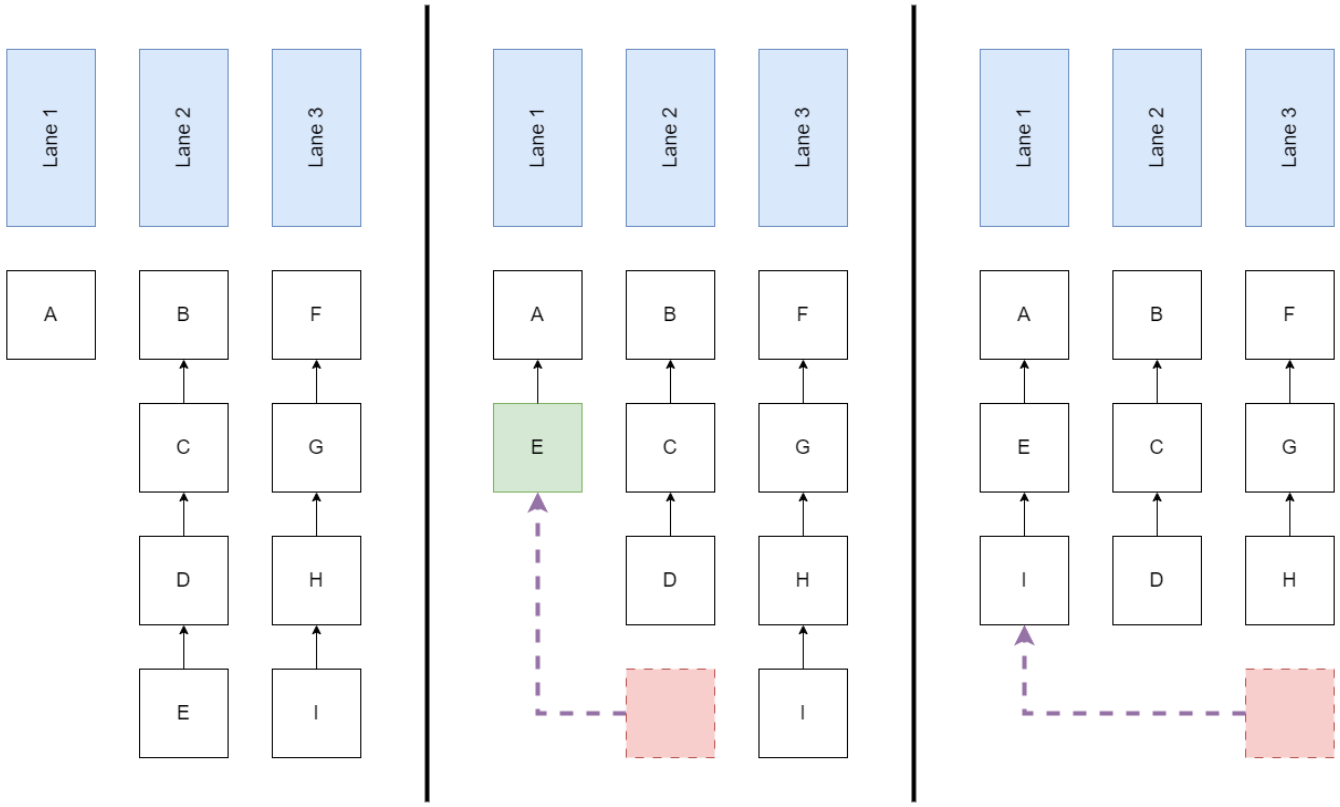
If multiple lanes have the same busyness, select the lane that comes first in the `CheckoutLane*` array.

If the difference between the MAX and MIN checkout lanes is ≤ 1, do nothing.

If there are less than 2 lanes, do nothing.

Return `true` if and only if a customer was moved; otherwise `false`.

The diagram below shows the resulting setup after two calls to a sample array of lanes.




```
int process_all_lanes(CheckoutLane* lanes[], int number_of_lanes)
```

Given an array of `CheckoutLane*`, `process()` each `CheckoutLane` a single time and return the the sum of the result.

```
void close_store(CheckoutLane* lanes[], int number_of_lanes)
```

It's closing time. Given an array of `CheckoutLane*`, free all memory associated with them. Any customers still left in the queue is kicked out and also freed from memory.

Final Remarks:

Use piazza and office hours wisely! Start early and work diligently. Assignments are more challenging than exercises, so you will need to work productively and not wait until the last minute.

Please submit the contents of your `wackystore.c`. The file name does not matter.