version: 1.0 last updated: 2025-07-09 09:00

CS Amusement Park

Overview

Welcome to CS: Amusement Park 🎡

In response to a growing demand for better theme park management, the Amusement Park Association has recruited you to create a program that designs and simulates the operations of a bustling amusement park to ensure all visitors leave with a smile.

Assignment Structure

This assignment will test your ability to create, use, manipulate and solve problems using linked lists. To do this, you will be implementing an amusement park simulator, where rides are represented as a linked list, stored within the park. Each ride will contain a list of visitors currently in the queue for that ride. The park will also contain a list of visitors who are currently roaming the park, and not in the queue for a specific ride.

Starter Code

This assignment utilises a multi-file system. There are three files we use in CS Amusement Park:

- **Main File (`main.c`):** This file contains the main function, which serves as the entry point for the program. It is responsible for displaying the welcome message, prompting the user for the park's name, and initiating the command loop that interacts with the park.

– Main File Walkthrough

A simple `main()` function has been provided in main.c to get you started. This function outlines the basic flow of the program, which you will build upon as you implement the amusement park's functionality.

```c
int main(void) {
    // Welcome message
    printf("Welcome to CS Amusement Park!\n");
    printf(
        "                                          o\n"
        "                                        o |\n"
        "                                        o |\n"
        "                                          |\n"
        "      .       .           ._._.    _                    .===.\n"
        "      |`      |`        ..'\\ /`.. |H|        .--.      .:'   `:.\n"
        "     //\\-...-/|\\         |- o -|  |H|`.     /||||\\     ||     ||\n"
        " ._.'//////,'|||`._.    '`./|\\.'` |\\\\\||:. .'||||||`.   `:.   .:'\n"
        " ||||||||||||[ ]||||      /_T_\\   |:`:.--'||||||||||`--..`=:='...\n\n"
    );

    printf("Enter the name of the park: ");
    char name[MAX_SIZE];
    scan_name(name);
    struct park *park = initialise_park(name);

    // Command loop
    command_loop(park);

    // End message
    printf("\nGoodbye!\n");

    return 0;
}
```

Printing a Welcome Message

The first statement in the `main()` function prints a welcome message, along with an ASCII Amusement Park. After displaying the welcome message, the program prompts the user to input a name for the amusement park.

```
printf("Welcome to CS Amusement Park!\n");
printf(
    "                                          o\n"
    "                                        o |\n"
    "                                        |\n"
    "      .       .           ._._.    _                    .===.\n"
    "      |`      |`        ..'\\ /`.. |H|        .--.      .:'   `:.\n"
    "     //\\-...-/|\\         |- o -|  |H|`.     /||||\\     ||     ||\n"
    " ._.'//////,'|||`._.    '`./|\\.'` |\\\\\||:. .'||||||`.   `:.   .:'\n"
    " ||||||||||||[ ]||||      /_T_\\   |:`:.--'||||||||||`--..`=:='...\n\n"
);
printf("Enter the name of the park: ");
```

Scanning in the Park Name

In this section, the program receives input from the user for the park's name:

- It creates a character array, name, of size `MAX_SIZE` to store the park's name.
- It utilises the provided `scan_name()` function, which reads the user's input and stores it in the name array.

```
char name[MAX_SIZE];
scan_name(name);
struct park *park = initialise_park(name);
```

Initialising the Park

Once the name of the park is acquired, the program proceeds to initialize the park itself:

- It calls the `initialise_park()` function, passing the `name` string as an argument. This function, which you will need to implement in stage 1.1, is responsible for setting up the initial state of the park, including allocating memory and setting default values.
- The function returns a pointer to a `struct park`, which represents the initialised park.

```
struct park *park = initialise_park(name);
```

Entering the Command Loop

After initialising the park, the program enters the command loop by calling the `command_loop()` function which you will need to implement in stage 1.2:

- This function will handle various user commands to interact with the park, such as adding rides, managing visitors, and printing the park's status.

Ending the Program

Finally, after the user decides to quit the program, a farewell message is printed:

```
printf("\nGoodbye!\n");
```

- **Header File (`cs_amusement_park.h`):** This file declares constants, enums, structs, and function prototypes for the program. **You will need to add prototypes for any functions you create in this file**. You should also add **any of your own constants and/or enums** to this file.
- **Implementation File (`cs_amusement_park.c`):** This file is where most of the assignment work will be done, as you will implement the logic and functionality required by the assignment here. This file also contains stubs of functions for stage 1 you to implement. It does not contain a `main` function, so you will need to compile it alongside `main.c`. **You will need to define any additional functions you may need for later stages in this file**.

> NOTE:
>
> **Function Stub**: A temporary substitute for yet-to-be implemented code. It is a placeholder function that shows what the function will look like, but does nothing currently.

The implementation file `cs_amusement_park.c` contains some provided functions to help simplify some stages of this assignment. These functions have been fully implemented for you and should not need to be modified to complete this assignment.

These provided functions will be explained in the relevant stages of this assignment. **Please read the function comments and the specification as we will suggest certain provided functions for you to use.**

> **NOTE:**
>
> If you wish to create your own helper functions, you can put the function prototypes in `cs_amusement_park.h` and implement the function in `cs_amusement_park.c`. You should place your function comment just above the function definition in `cs_amusement_park.c`, similar to the provided functions.

We have defined some structs in `cs_amusement_park.h` to get you started. You may add fields to any of the structs if you wish.

– Structs

`struct park`

- **Purpose**: To store all the information about the amusement park.
  It contains:
  - The name of the park.
  - The total number of visitors in the park.
  - A list of rides within the park.
  - A list of visitors currently roaming the park.

`struct ride`

- **Purpose**: To represent a ride within the amusement park.
  It includes:
  - The name of the ride.
  - The type of the ride (e.g., Roller Coaster, Carousel, Ferris Wheel or Bumper Cars).
  - The maximum number of riders it can accommodate.
  - The maximum number of visitors that can wait in the queue.
  - The minimum height requirement for the ride.
  - A queue of visitors waiting for the ride.
  - A pointer to the next ride in the park.

`struct visitor`

- **Purpose**: To represent a visitor in the amusement park.
  It contains:
  - The visitor's name.
  - The visitor's height.
  - A pointer to the next visitor in the park or in a ride queue.

The following enum definition is also provided for you in `cs_amusement_park.h`. You can create your own enums if you would like, but you should not modify the provided enums.

– Enums

`enum ride_type`

- **Purpose**: To represent the type of ride within the amusement park.
  The possible types are:
  - `ROLLER_COASTER`,
  - `CAROUSEL`,
  - `FERRIS_WHEEL`,
  - `BUMPER_CARS`,
  - `INVALID`.

> **HINT:**
>
> Remember to initalise every field inside the structs when creating them (not just the fields you are using at that moment).

Getting Started

There are a few steps to getting started with CS Amusement Park.

1. Create a new folder for your assignment work and move into it.

```
$ mkdir ass2
$ cd ass2
```

2. Use this command on your CSE account to copy the file into your current directory:

```
$ 1091 fetch-activity cs_amusement_park
```

3. Run `1091 autotest cs_amusement_park` to make sure you have correctly downloaded the file.

```
$ 1091 autotest cs_amusement_park
```

> **NOTE:**
>
> When running the autotest on the starter code (with no modifications), it is expected to see failed tests as there is still very little code to make up the assignment!

4. Read through **Stage 1**.

5. Spend a few minutes playing with the reference solution -- get a feel for how the assignment works.

```
$ 1091 cs_amusement_park
```

5. Think about your solution, draw some diagrams to help you get started.

6. Start coding!

Reference Implementation

To help you understand the proper behaviour of the Amusement Park Simulator, we have provided a reference implementation. If you have any questions about the behaviour of your assignment, you can check and compare it to the reference implementation.

To run the reference implementation, use the following command:

```
$ 1091 cs_amusement_park
```

You might want to start by running the `?` command:

– Example Usage

```
$ 1091 cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                      o |
                                        |
      .       .           ._._.    _                    .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]|||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW!
Enter command: ?
======================[ CS Amusement Park ]======================
      ===============[     Usage Info     ]===============
  a r [ride_name] [ride_type]
    Add a ride to the park.
  a v [visitor_name] [visitor_height]
    Add a visitor to the park.
  i [index] [ride_name] [ride_type]
    Insert a ride at a specific position in the park's ride list.
  j [ride_name] [visitor_name]
    Add a visitor to the queue of a specific ride.
  m [visitor_name] [ride_name]
    Move a visitor from roaming to a ride's queue.
  d [visitor_name]
    Remove a visitor from any ride queue and return to roaming.
  p
    Print the current state of the park, including rides and
    visitors.
  t
    Display the total number of visitors in the park, including
    those roaming and in ride queues.
  c [start_ride] [end_ride]
    Count and display the number of visitors in queues between
    the specified start and end rides, inclusive.
  l [visitor_name]
    Remove a visitor entirely from the park.
  r
    Operate all rides, allowing visitors to enjoy the rides
    and moving them to roaming after their ride.
  M [ride_type]
    Merge the two smallest rides of the specified type.
  s [n] [ride_name]
    Split an existing ride into `n` smaller rides.
  q
    Quit the program and free all allocated resources.
  ?
    Show this help information.
================================================================
```

Allowed C Features

In this assignment, there are no restrictions on C Features, except for those in the Style Guide. If you choose to disregard this advice, you must still follow the Style Guide.

You also may be unable to get help from course staff if you use features not taught in DPST1091. Features that the Style Guide identifies as illegal will result in a penalty during marking. You can find the style marking rubric above. **Please note that this assignment must be completed using only Linked Lists. You may only use strings for their intended purpose: storing names such as those of parks, rides, or visitors. This includes reading, writing, and storing these names. You must not use arrays of any kind to assist with solving the logic of the assignment. Arrays must not be used to store, manipulate, or process data beyond string names. Doing so will result in a zero for performance. All core data structures and logic must be implemented using linked lists only.**

FAQ

+ FAQ

NOTE:

You can assume that your program will **NEVER** be given:

- Command arguments that are not of the right type.
- An incorrect number of arguments for the specific command.

Additionally, commands will always start with a `char` .

Stages

The assignment has been divided into incremental stages.

– Stage List

Stage 1

- Stage 1.1 - Creating the Park.
- Stage 1.2 - Command Loop and Help.
- Stage 1.3 - Append Rides and Visitors.
- Stage 1.4 - Printing the Park.
- Stage 1.5 - Handling Errors.

Stage 2

- Stage 2.1 - Inserting New Rides.
- Stage 2.2 - Add Visitors to the Queue of Rides.
- Stage 2.3 - Remove Visitors from the Queue of Rides.
- Stage 2.4 - Move Visitors to Different Rides.
- Stage 2.5 - Total Visitors.

Stage 3

- Stage 3.1 - End of Day.
- Stage 3.2 - Leave the Park.
- Stage 3.3 - Operate the Rides.
- Stage 3.4 - Ride Shut Down.

Stage 4

- Stage 4.1 - Merge.
- Stage 4.2 - Split.
- Stage 4.3 - Scheduled.

Extension (not for marks)

- SplashKit - Create and Share a Graphical User Interface.

Your Tasks

This assignment consists of four stages. Each stage builds on the work of the previous stage, and each stage has a higher complexity than its predecessor. You should complete the stages in order.

Stage 1 ●○○ | Stage 2 ●○○ | Stage 3 ●●○ | Stage 4 ●●● | Extension | Tools

Stage 2

For **Stage 2** of this assignment, you will use some more advanced linked list knowledge to modify linked list nodes, and perform calculations based on information stored over several nodes. This milestone has been divided into five substages:

- Stage 2.1 - Inserting New Rides.
- Stage 2.2 - Add Visitors to the Queue of Rides.
- Stage 2.3 - Remove Visitors from the Queue of Rides.
- Stage 2.4 - Move Visitors to Different Rides.
- Stage 2.5 - Total Visitors.

> **NOTE:**
>
> From this stage onwards, there are no provided function stubs for the remaining stages. It is up to you to create your own functions as needed to implement the desired functionality for the assignment. Be sure to put your function prototypes in `cs_amusement_park.h` and your function definitions in `cs_amusement_park.c`.

Stage 2.1 - Inserting New Rides

Currently, we can only extend our park and add new rides at the end of the list. You want to be able to customise the order of rides a little more, resulting in the need for a way to insert rides anywhere.

Command

```
i [n] [name] [type]
```

Description

The `i` command is similar to `a` (append ride), except that it takes in an additional integer argument `n` which is the position in the park at which to insert the new ride.

It should create a new `struct ride` containing the `ride_name` and `ride_type` and insert it into the park at position `n`.

For example:

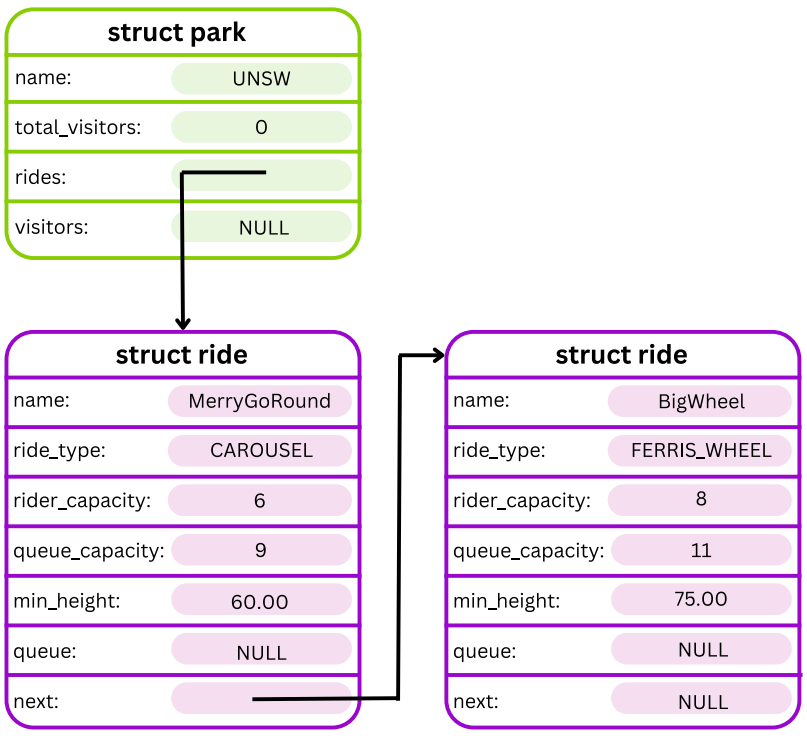
- If `n == 1`,
  then the new ride should be inserted at the *head* of the park.
- If `n == 2`,
  then the new ride should be inserted after the first ride in the park.
- If `n == N`,
  then the new ride should be inserted after the `N` th ride in the park.
- If `n` is greater than the length of the park,
  then the new ride should be inserted at the tail of the linked list.

After successful insertion, your program should print the following message:

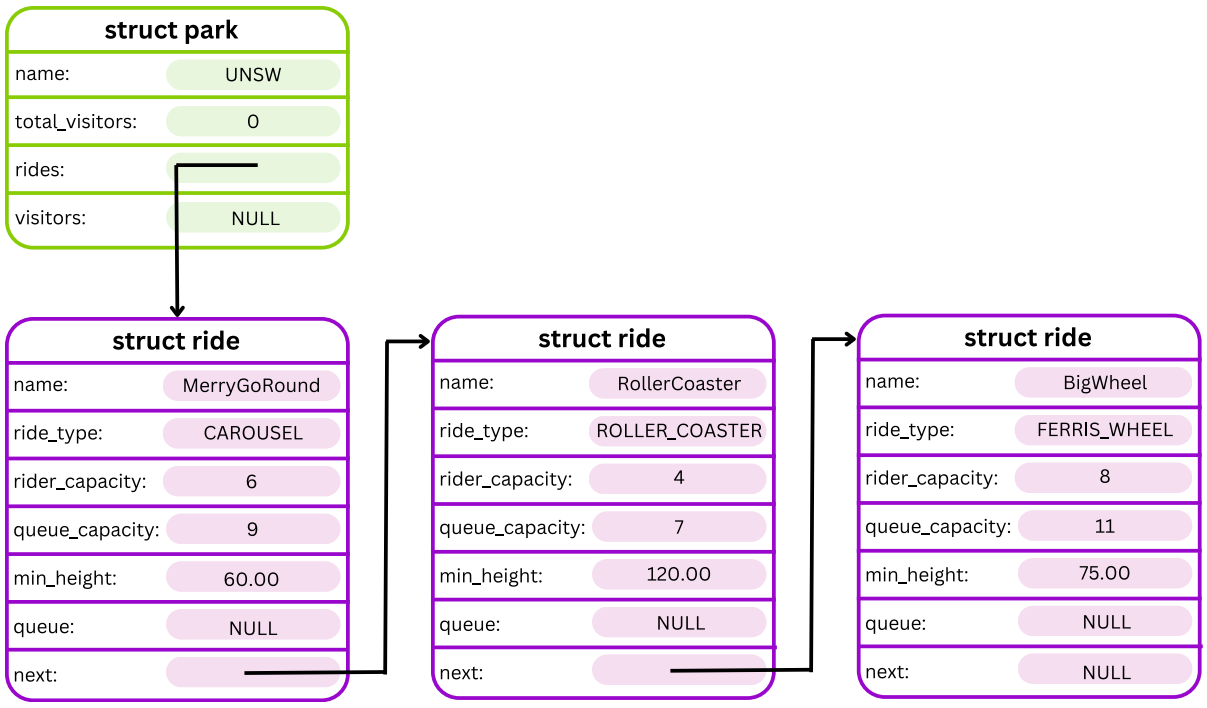
`Ride: '[name]' inserted!`

Inserting a new ride at position `n == 2` should look something like this:

Before Inserting:



After Inserting:



Errors

Just like command `a` : **append ride**, there are restrictions on the rides that can be inserted into the park.
Specifically, if one of the following conditions is met, then the ride should NOT be inserted, and an error message should be printed out instead.

- If `n < 1` , the following error should be printed:
  `ERROR: n must be at least 1.`
- If `type` is `INVALID` , the following error should be printed:
  `ERROR: Invalid ride type.`
- If `ride_name` already exists within the park, the following error should be printed:
  `ERROR: '[name]' already exists.`

Examples

– Example 2.1.1: Insert ride at head

Input:

```
UNSW
a r MerryGoRound CAROUSEL
a r BigWheel FERRIS_WHEEL
i 1 RollerCoaster ROLLER_COASTER
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                       o |
                                       |
      .       .           ._._.    _                    .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]|||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a r MerryGoRound CAROUSEL
Ride: 'MerryGoRound' added!
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: i 1 RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' inserted!
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  MerryGoRound (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      No visitors
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
Visitors:
  No visitors!

Enter command: [CTRL+D]
Goodbye!
```

- Example 2.1.2: Insert ride at tail

Input:

```
UNSW
a r MerryGoRound CAROUSEL
a r BigWheel FERRIS_WHEEL
i 5 RollerCoaster ROLLER_COASTER
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                       o |
                                       |
      .       .           ._._.    _                    .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a r MerryGoRound CAROUSEL
Ride: 'MerryGoRound' added!
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: i 5 RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' inserted!
Enter command: p
==================[ UNSW ]==================
Rides:
  MerryGoRound (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      No visitors
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
Visitors:
  No visitors!

Enter command: [CTRL+D]
Goodbye!
```

- Example 2.1.3: Invalid insertion

Input:

```
UNSW
i 0 RollerCoaster ROLLER_COASTER
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                       o |
                                       |
      .       .           ._._.    _                    .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]|||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: i 0 RollerCoaster ROLLER_COASTER
ERROR: n must be at least 1.
Enter command: p
==================[ UNSW ]==================
The amusement park is empty!

Enter command: [CTRL+D]
Goodbye!
```

Clarifications

- If `n` is 1, the new ride should be inserted at the head of the linked list.
- If `n` is greater or equal to the length of the linked list, then the new ride should be inserted at the end of the linked list.
- When checking if a ride name already exists in the park, the comparison is case-sensitive. For example, if a ride named `DPST1091` is already in the park, trying to add a ride named `Dpst1091` will not be treated as a duplicate, and no error will be raised.

> **NOTE:**
>
> You may like to autotest this section with the following command:
>
> ```
> 1091 autotest-stage 02_01 cs_amusement_park
> ```

Stage 2.2 - Add Visitors to the Queue of Rides

Now that we have built an amusement park with rides, it's time to start letting visitors join the queues for those rides and have some fun!

Command

```
j [ride_name] [visitor_name]
```

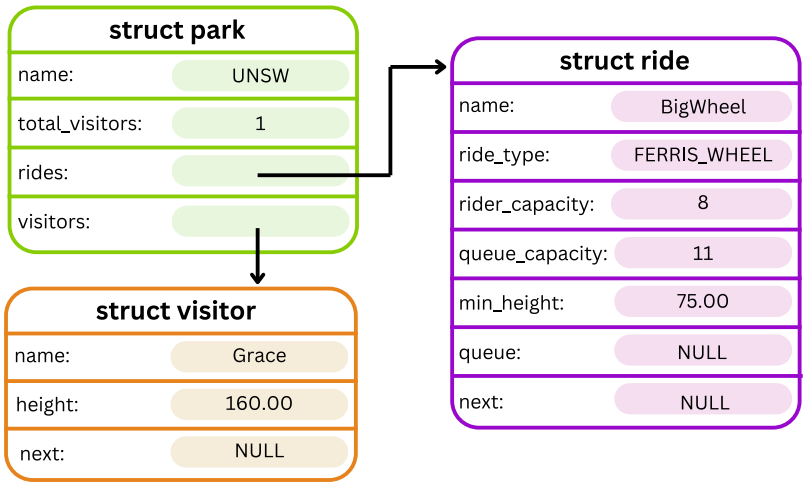
Description

The `j` command takes in two strings `ride_name` and `visitor_name` .

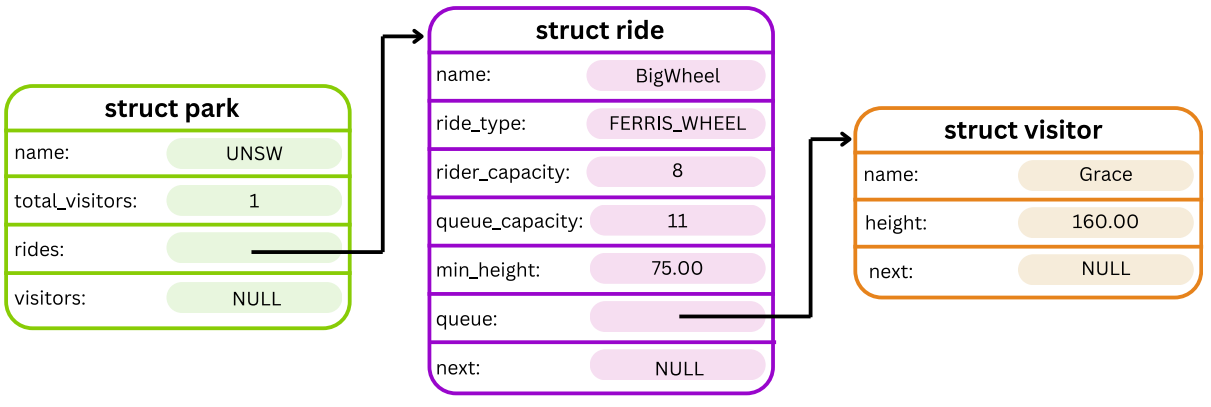
The `j` command should attempt to find the ride with a name matching `ride_name` , and then append the visitor currently roaming the park with name `visitor_name` to its ride queue.

On success, after moving a visitor into a ride queue the program should print `Visitor: '[visitor_name]' has entered the queue for '[ride_name]'.`

If we have one visitor and one ride in the park, before adding the visitor to the queue for the ride the park will look something like this:



After moving the visitor into the queue for the ride with the `j` command the park will look something like this:



Errors

If one of the following errors occurs, then the visitor should not join the queue of the ride, and an error message should be printed instead.

- If no ride containing `ride_name` exists, the following error should be printed:

  `ERROR: No ride exists with name '[ride_name]'.`
- If no visitor containing `visitor_name` exists, the following error should be printed:

  `ERROR: No visitor exists with name '[visitor_name]'.`
- If the visitor does not meet the height requirement for the ride, the following error should be printed

  `ERROR: '[visitor_name]' is not tall enough to ride '[ride_name]'.`
- If the queue is already at capacity, the following error should be printed

  `ERROR: The queue for '[ride_name]' is full. '[visitor_name]' cannot join the queue.`

Examples

– Example 2.2.1: Successfully add visitor to ride

Input:

```
UNSW
a r BigWheel FERRIS_WHEEL
a v Sofia 160
j BigWheel Sofia
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                      o |
                                        |
      .       .           ._._.    _                    .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a v Sofia 160
Visitor: 'Sofia' has entered the amusement park!
Enter command: j BigWheel Sofia
Visitor: 'Sofia' has entered the queue for 'BigWheel'.
Enter command: p
==================[ UNSW ]==================
Rides:
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      Sofia (160.00cm)
Visitors:
  No visitors!

Enter command: [CTRL+D]
Goodbye!
```

– Example 2.2.2: Error adding a visitor to ride #1

Input:

```
UNSW
j RollerCoaster Sofia
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                      o |
                                        |
      .       .           ._._.    _                    .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: j RollerCoaster Sofia
ERROR: No ride exists with name 'RollerCoaster'.
Enter command: p
==================[ UNSW ]==================
The amusement park is empty!

Enter command: [CTRL+D]
Goodbye!
```

- Example 2.2.3: Error adding a visitor to ride #2

Input:

```
UNSW
a r BigWheel FERRIS_WHEEL
a v Sasha 70
j BigWheel Sasha
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                      o |
                                        |
      .       .           ._._.    _                      .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.     '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]|||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a v Sasha 70
Visitor: 'Sasha' has entered the amusement park!
Enter command: j BigWheel Sasha
ERROR: 'Sasha' is not tall enough to ride 'BigWheel'.
Enter command: p
==================[ UNSW ]==================
Rides:
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
Visitors:
  Sasha (70.00cm)

Enter command: [CTRL+D]
Goodbye!
```

Clarifications

- If more than one error occurs, **only the first error in the order specified above** should be addressed by printing an error message. This is the same for all future commands.
- When checking a name, case matters. For example, the name `DPST1091`, does not match the name `Dpst1091`. This is the same for all future commands
- The `j` command only allows a visitor to enter a ride queue if they are currently roaming in the park.

> **NOTE:**
>
> You may like to autotest this section with the following command:
>
> ```
> 1091 autotest-stage 02_02 cs_amusement_park
> ```

Stage 2.3 - Remove Visitors from the Queue of Rides Command

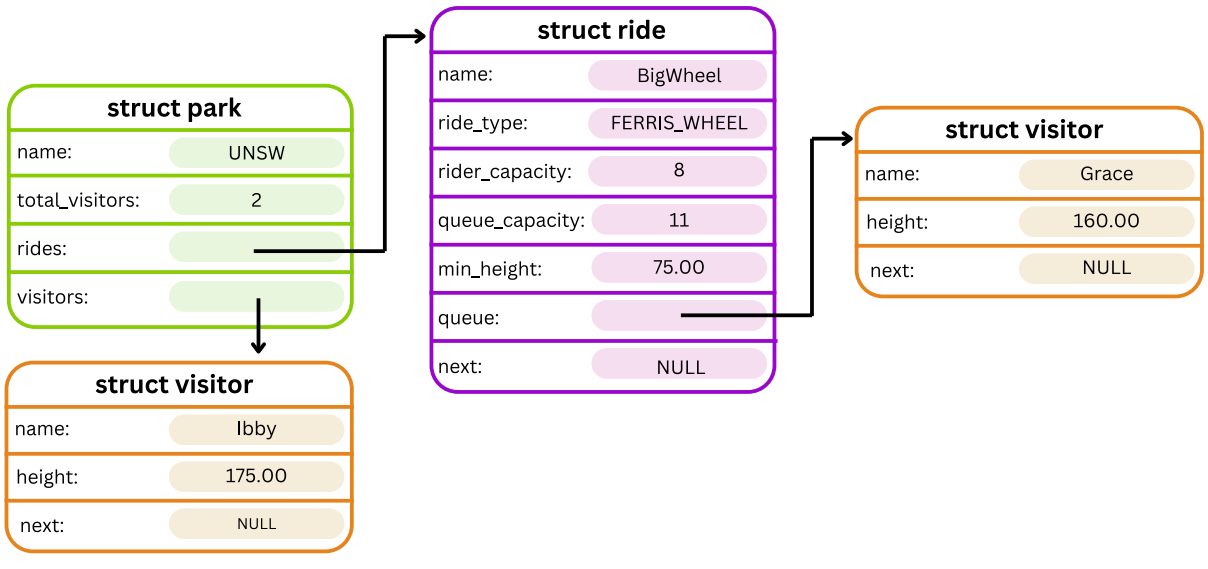
```
d [passenger_name]
```

Description

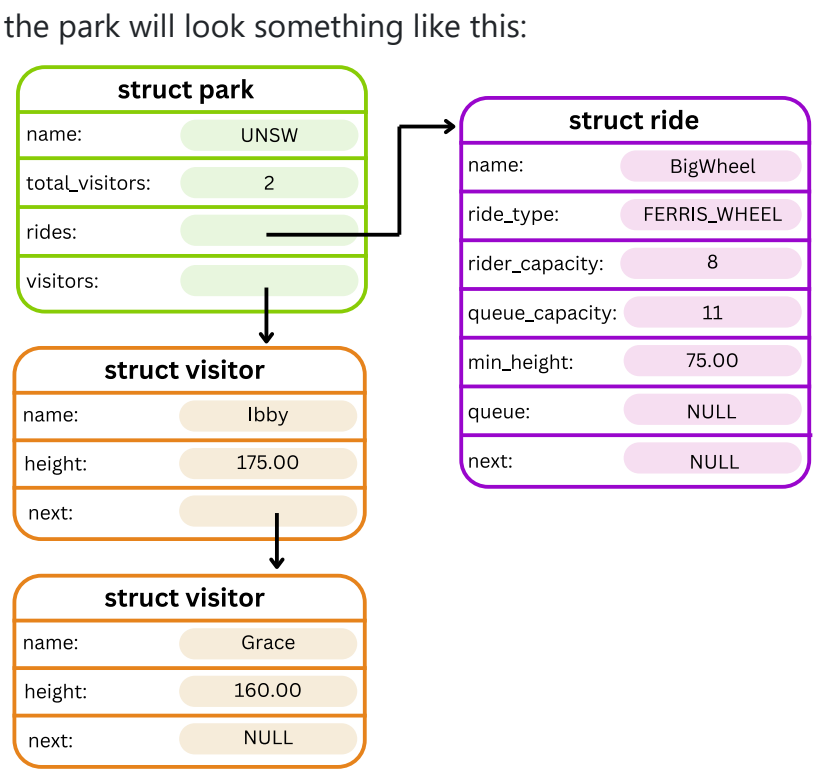
The `d` command should attempt to find the visitor with name `visitor_name` in a queue for a ride and remove them from the queue, returning them to the park, i.e. appending them to the tail of the park visitor list.

It should then print: `Visitor: '[visitor_name]' has been removed from their ride queue and is now roaming the park.`

Before removing the visitor Grace from the queue of the ride, the park will look something like this:



After running the command `d Grace` the park will look something like this:



Errors

If one of the following errors occur, then the visitor should not be removed from the queue of a ride, and an error message should be printed instead.

- If no visitor containing `visitor_name` exists in a queue, the following error should be printed:
  `ERROR: Visitor '[visitor_name]' not found in any queue.`

Examples

– Example 2.3.1: Remove visitor from queue

Input:

```
UNSW
a r BigWheel FERRIS_WHEEL
a v Sofia 160
j BigWheel Sofia
d Sofia
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                       o |
                                       |
      .       .           ._._.    _                    .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a v Sofia 160
Visitor: 'Sofia' has entered the amusement park!
Enter command: j BigWheel Sofia
Visitor: 'Sofia' has entered the queue for 'BigWheel'.
Enter command: d Sofia
Visitor: 'Sofia' has been removed from their ride queue and is now roaming the park.
Enter command: p
==================[ UNSW ]==================
Rides:
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
Visitors:
  Sofia (160.00cm)

Enter command: [CTRL+D]
Goodbye!
```

- Example 2.3.2: Error removing visitor from queue

Input:

```
UNSW
d Sasha
a r BigWheel FERRIS_WHEEL
a v Ibby 160
d Ibby
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                       o |
                                       |
      .       .           ._._.    _                      .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: d Sasha
ERROR: Visitor 'Sasha' not found in any queue.
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a v Ibby 160
Visitor: 'Ibby' has entered the amusement park!
Enter command: d Ibby
ERROR: Visitor 'Ibby' not found in any queue.
Enter command: p
==================[ UNSW ]==================
Rides:
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
Visitors:
  Ibby (160.00cm)

Enter command: [CTRL+D]
Goodbye!
```

– Example 2.3.3: Remove multiple visitors from a queue

Input:

```
UNSW
a r BigWheel FERRIS_WHEEL
a v Sofia 160
j BigWheel Sofia
a v Sasha 150
a v Grace 165
a v Ibby 175
j BigWheel Grace
j BigWheel Sasha
p
d Grace
d Sofia
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                      o |
                                      |
      .       .           ._._.    _                    .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]|||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a v Sofia 160
Visitor: 'Sofia' has entered the amusement park!
Enter command: j BigWheel Sofia
Visitor: 'Sofia' has entered the queue for 'BigWheel'.
Enter command: a v Sasha 150
Visitor: 'Sasha' has entered the amusement park!
Enter command: a v Grace 165
Visitor: 'Grace' has entered the amusement park!
Enter command: a v Ibby 175
Visitor: 'Ibby' has entered the amusement park!
Enter command: j BigWheel Grace
Visitor: 'Grace' has entered the queue for 'BigWheel'.
Enter command: j BigWheel Sasha
Visitor: 'Sasha' has entered the queue for 'BigWheel'.
Enter command: p
==================[ UNSW ]==================
Rides:
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      Sofia (160.00cm)
      Grace (165.00cm)
      Sasha (150.00cm)
Visitors:
  Ibby (175.00cm)

Enter command: d Grace
Visitor: 'Grace' has been removed from their ride queue and is now roaming the park.
Enter command: d Sofia
Visitor: 'Sofia' has been removed from their ride queue and is now roaming the park.
Enter command: p
==================[ UNSW ]==================
Rides:
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      Sasha (150.00cm)
Visitors:
  Ibby (175.00cm)
  Grace (165.00cm)
  Sofia (160.00cm)

Enter command: [CTRL+D]
Goodbye!
```

Clarifications

- If more than one error occurs, **only the first error in the order specified above** should be addressed by printing an error message. This is the same for all future commands.

- When checking a name, case matters. For example, the name `DPST1091`, does not match the name `Dpst1091`. This is the same for all future commands

> **NOTE:**
>
> You may like to autotest this section with the following command:
>
> ```
> 1091 autotest-stage 02_03 cs_amusement_park
> ```

Stage 2.4 - Move Visitors to Different Rides

Command

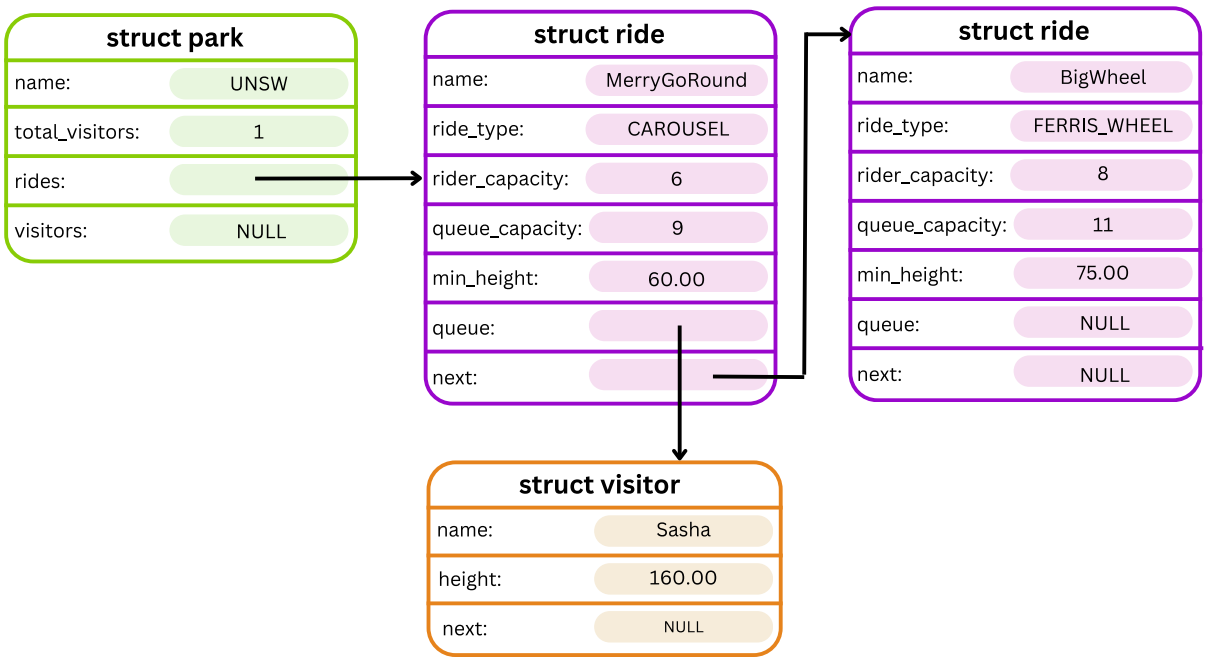
```
m [visitor_name] [ride_name]
```

Description

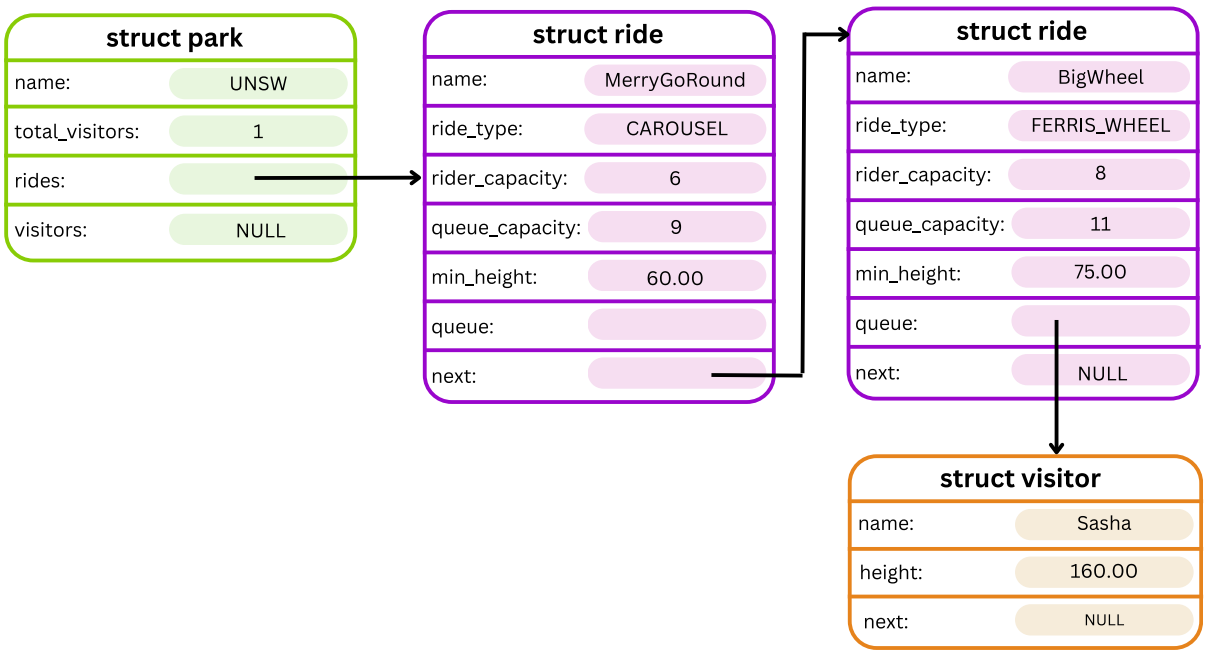
The `m` command should attempt to find the visitor in the park and move them to the queue for `ride_name`.
The visitor may be roaming the park or in the queue for any ride.

On success, after moving a visitor into the ride queue the program should print `Visitor: '[visitor_name]' has been moved to the queue for '[ride_name]'.`

Before removing the visitor Sasha, the park will look something like this:



After running the command `m Sasha BigWheel` the park will look something like this:



Errors

If one of the following errors occur, then the visitor should not be moved, and an error message should be printed instead.

- If a ride with name `ride_name` does not exist within the park, the following error should be printed:
  `ERROR: No ride exists with name '[ride_name]'.`
- If a visitor with name `visitor_name` does not exist in the park, nor in a queue for a ride in the park, the following error should be printed:
  `ERROR: No visitor with name: '[visitor_name]' exists.`
- If the visitor is currently in the queue for the ride they are trying to move into, the following error should be printed:
  `ERROR: '[visitor_name]' is already in the queue for '[ride_name]'.` .
- If the visitor does not meet the height requirement for the ride, the following error should be printed
  `ERROR: '[visitor_name]' is not tall enough to ride '[ride_name]'.`

- If the ride queue is already at capacity, the following error should be printed

  ```
  ERROR: The queue for '[ride_name]' is full. '[visitor_name]' cannot join the queue.
  ```

Examples

– Example 2.4.1: Move visitor to another queue

Input:

```
UNSW
a r BigWheel FERRIS_WHEEL
a r MerryGoRound CAROUSEL
a v Sofia 160
j BigWheel Sofia
m Sofia MerryGoRound
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                      o |
                                      |
      .       .           ._._.    _                    .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a r MerryGoRound CAROUSEL
Ride: 'MerryGoRound' added!
Enter command: a v Sofia 160
Visitor: 'Sofia' has entered the amusement park!
Enter command: j BigWheel Sofia
Visitor: 'Sofia' has entered the queue for 'BigWheel'.
Enter command: m Sofia MerryGoRound
Visitor: 'Sofia' has been moved to the queue for 'MerryGoRound'.
Enter command: p
==================[ UNSW ]==================
Rides:
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
  MerryGoRound (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      Sofia (160.00cm)
Visitors:
  No visitors!

Enter command: [CTRL+D]
Goodbye!
```

– Example 2.4.2: Visitor already in the queue

Input:

```
UNSW
a r BigWheel FERRIS_WHEEL
a v Grace 160
j BigWheel Grace
m Grace BigWheel
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                      o |
                                        |
      .       .           ._._.    _                    .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a v Grace 160
Visitor: 'Grace' has entered the amusement park!
Enter command: j BigWheel Grace
Visitor: 'Grace' has entered the queue for 'BigWheel'.
Enter command: m Grace BigWheel
ERROR: 'Grace' is already in the queue for 'BigWheel'.
Enter command: [CTRL+D]
Goodbye!
```

– Example 2.4.3: Target ride does not exist

Input:

```
UNSW
a r BigWheel FERRIS_WHEEL
a r MerryGoRound CAROUSEL
a v Sasha 160
j BigWheel Sofia
m Sasha NonExistent
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                      o |
                                        |
      .       .           ._._.    _                      .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a r MerryGoRound CAROUSEL
Ride: 'MerryGoRound' added!
Enter command: a v Sasha 160
Visitor: 'Sasha' has entered the amusement park!
Enter command: j BigWheel Sofia
ERROR: No visitor exists with name 'Sofia'.
Enter command: m Sasha NonExistent
ERROR: No ride exists with name 'NonExistent'.
Enter command: p
==================[ UNSW ]==================
Rides:
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
  MerryGoRound (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      No visitors
Visitors:
  Sasha (160.00cm)

Enter command: [CTRL+D]
Goodbye!
```

Clarifications

- A visitor can be moved either from a ride's queue or from roaming the park.
- Note, you may need to go back and modify your error checking for adding a visitor to the park to account for visitors in the queue of rides.

> **NOTE:**
>
> You may like to autotest this section with the following command:
>
> ```
> 1091 autotest-stage 02_04 cs_amusement_park
> ```

Stage 2.5 - Total Visitors

This stage involves implementing two very similar commands. You can implement them in any order.

Command 1

```
t
```

Description

Print out the total number of visitors in the park, this includes all visitors in queues and roaming the park. You should print in the following form:

```
Total visitors: [total_visitors]
Visitors walking around: [roaming_visitors]
Visitors in queues: [queued_visitors]
```

Examples

- Example 2.5.1: Total visitors

Input:

```
UNSW
a v Sofia 160
a v Sasha 150
t
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                      o |
                                      |
                                                                ===
                                      |
      .       .          ._._.    _                          .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a v Sofia 160
Visitor: 'Sofia' has entered the amusement park!
Enter command: a v Sasha 150
Visitor: 'Sasha' has entered the amusement park!
Enter command: t
Total visitors: 2
Visitors walking around: 2
Visitors in queues: 0
Enter command: [CTRL+D]
Goodbye!
```

Command 2

```
c [start_name] [end_name] [direction]
```

Description

The `c` command should attempt to find the ride with `ride_name` matching `start_name`, and the ride with `ride_name` matching `end_name`.

It should then count the total number of visitors, waiting in the queue in the specified `direction`.

- If `direction` is `>` it should then count the total number of visitors, from ride: `start_name` to ride: `end_name`, inclusive, wrapping if needed.
- If `direction` is `<` it should then count the total number of visitors, from ride: `end_name` to ride: `start_name`, inclusive, wrapping if needed.

It should print out the final result in the following format:

- If `direction` is `>`:

```
Total visitors from '[start_name]' to '[end_name]': [range_visitors].
```

- If `direction` is `<`:

```
Total visitors from '[end_name]' to '[start_name]': [range_visitors].
```

Errors

- If either `start_name` or `end_name` do not exist, the following error should be printed:
  `ERROR: One or both rides do not exist ('[start_name]' or '[end_name]').`

Examples

- Example 2.5.2: Count visitors between rides forward

Input:

```
UNSW
a r BigWheel FERRIS_WHEEL
a r MerryGoRound CAROUSEL
a v Grace 165
a v Ibby 170
j BigWheel Grace
j MerryGoRound Ibby
a v Sasha 160.5
a v Sofia 162
c BigWheel MerryGoRound >
t
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                      o |
                                        |
      .       .           ._._.    _                      .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.    '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a r MerryGoRound CAROUSEL
Ride: 'MerryGoRound' added!
Enter command: a v Grace 165
Visitor: 'Grace' has entered the amusement park!
Enter command: a v Ibby 170
Visitor: 'Ibby' has entered the amusement park!
Enter command: j BigWheel Grace
Visitor: 'Grace' has entered the queue for 'BigWheel'.
Enter command: j MerryGoRound Ibby
Visitor: 'Ibby' has entered the queue for 'MerryGoRound'.
Enter command: a v Sasha 160.5
Visitor: 'Sasha' has entered the amusement park!
Enter command: a v Sofia 162
Visitor: 'Sofia' has entered the amusement park!
Enter command: c BigWheel MerryGoRound >
Total visitors from 'BigWheel' to 'MerryGoRound': 2.
Enter command: t
Total visitors: 4
Visitors walking around: 2
Visitors in queues: 2
Enter command: [CTRL+D]
Goodbye!
```

- Example 2.5.3: Count visitors between rides backwards

Input:

```
UNSW
a r BigWheel FERRIS_WHEEL
a r MerryGoRound CAROUSEL
a v Grace 165
a v Ibby 170
i 1 RollerCoaster ROLLER_COASTER
j BigWheel Grace
j MerryGoRound Ibby
a v Sasha 160.5
a v Sofia 162
c RollerCoaster MerryGoRound <
t
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
                                        o
                                      o |
                                      |
      .       .           ._._.    _                    .===.
      |`      |`        ..'\ /`.. |H|        .--.      .:'   `:.
     //\-...-/|\         |- o -|  |H|`.     /||||\     ||     ||
 ._.'//////,'|||`._.     '`./|\.'` |\\||:. .'||||||`.   `:.   .:'
 ||||||||||||[ ]||||      /_T_\   |:`:.--'||||||||||`--..`=:='...

Enter the name of the park: UNSW
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a r MerryGoRound CAROUSEL
Ride: 'MerryGoRound' added!
Enter command: a v Grace 165
Visitor: 'Grace' has entered the amusement park!
Enter command: a v Ibby 170
Visitor: 'Ibby' has entered the amusement park!
Enter command: i 1 RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' inserted!
Enter command: j BigWheel Grace
Visitor: 'Grace' has entered the queue for 'BigWheel'.
Enter command: j MerryGoRound Ibby
Visitor: 'Ibby' has entered the queue for 'MerryGoRound'.
Enter command: a v Sasha 160.5
Visitor: 'Sasha' has entered the amusement park!
Enter command: a v Sofia 162
Visitor: 'Sofia' has entered the amusement park!
Enter command: c RollerCoaster MerryGoRound <
Total visitors from 'MerryGoRound' to 'RollerCoaster': 1.
Enter command: t
Total visitors: 4
Visitors walking around: 2
Visitors in queues: 2
Enter command: [CTRL+D]
Goodbye!
```

Clarifications

- If `start_name` is the same as `end_name`, then count the queue for that particular ride only.
- `start_name` and `end_name` should be included in the range.
- If both `start_name` and `end_name` are invalid, the `start_name` should be printed in the error message.

> **NOTE:**
>
> You may like to autotest this section with the following command:
>
> ```
> 1091 autotest-stage 02_05 cs_amusement_park
> ```

Testing and Submission

Remember to do your own testing

Are you finished with this stage? If so, you should make sure to do the following:

- Run `1091 style` and clean up any issues a human may have reading your code. Don't forget -- **20%** of your mark in the assignment is based on style and readability!
- Autotest for this stage of the assignment by running the `autotest-stage` command as shown below.
- Remember -- *give early and give often*. Only your last submission counts, but why not be safe and submit right now?

```
$ 1091 style cs_amusement_park.c
$ 1091 style cs_amusement_park.h
$ 1091 style main.c
$ 1091 autotest-stage 02 cs_amusement_park
$ give dp1091 ass2_cs_amusement_park cs_amusement_park.c cs_amusement_park.h main.c
```

Assessment

Assignment Conditions

- **Joint work** is **not permitted** on this assignment.

  This is an individual assignment.

  The work you submit must be entirely your own work. Submission of any work even partly written by any other person is not permitted.

  The only exception being if you use small amounts (< 10 lines) of general purpose code (not specific to the assignment) obtained from a site such as Stack Overflow or other publicly available resources. You should attribute the source of this code clearly in an accompanying comment.

  Assignment submissions will be examined, both automatically and manually for work written by others.

  Do not request help from anyone other than the teaching staff of DPST1091/CPTG1391.

  Do not post your assignment code to the course forum - the teaching staff can view assignment code you have recently autotested or submitted with give.

  Rationale: this assignment is an individual piece of work. It is designed to develop the skills needed to produce an entire working program. Using code written by or taken from other people will stop you learning these skills.

- The use of **code-synthesis tools**, such as **GitHub Copilot**, is **not permitted** on this assignment.

  The use of **Generative AI** to generate code solutions is not permitted on this assignment.

  Rationale: this assignment is intended to develop your understanding of basic concepts. Using synthesis tools will stop you learning these fundamental concepts.

- **Sharing, publishing, distributing** your assignment work is **not permitted**.

  Do not provide or show your assignment work to any other person, other than the teaching staff of DPST1091/CPTG1391. For example, do not share your work with friends.

  Do not publish your assignment code via the internet. For example, do not place your assignment in a public GitHub repository.

  Rationale: by publishing or sharing your work you are facilitating other students to use your work, which is not permitted. If they submit your work, you may become involved in an academic integrity investigation.

- **Sharing, publishing, distributing your assignment work after the completion of DPST1091/CPTG1391** is **not permitted**.

  For example, do not place your assignment in a public GitHub repository after DPST1091/CPTG1391 is over.

  Rationale:DPST1091/CPTG1391 sometimes reuses assignment themes, using similar concepts and content. If students in future terms can find your code and use it, which is not permitted, you may become involved in an academic integrity investigation.

Violation of the above conditions may result in an academic integrity investigation with possible penalties, up to and including a mark of 0 in DPST1091/CPTG1391 and exclusion from UNSW.

Relevant scholarship authorities will be informed if students holding scholarships are involved in an incident of plagiarism or other misconduct. If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted - you may be penalised, even if the work was submitted without your knowledge or consent. This may apply even if your work is submitted by a third party unknown to you.

Submission of Work

You should submit intermediate versions of your assignment. Every time you autotest or submit, a copy will be saved as a backup. You can find those backups here, by logging in, and choosing the yellow button next to `ass2_cs_amusement_park`.

Every time you work on the assignment and make some progress, you should copy your work to your CSE account and submit it using the `give` command below.

It is fine if intermediate versions do not compile or otherwise fail submission tests.

Only the final submitted version of your assignment will be marked.

You submit your work like this:

```
$ give dp1091 ass2_cs_amusement_park cs_amusement_park.c cs_amusement_park.h main.c
```

Assessment Scheme

This assignment will contribute 25% to your final mark.

80% of the marks for this assignment will be based on the performance of the code you write in `cs_amusement_park.c cs_amusement_park.h main.c`.

20% of the marks for this assignment will come from manual marking of the readability of the C you have written. The manual marking will involve checking your code for clarity, and readability, which includes the use of functions and efficient use of loops and if statements.

Marks for your performance will be allocated roughly according to the below scheme.

100% for Performance	Completely Working Implementation, which exactly follows the specification (Stage 1, 2, 3 and 4).
85% for Performance	Completely working implementation of Stage 1, 2 and 3.
65% for Performance	Completely working implementation of Stage 1 and Stage 2.
35% for Performance	Completely working implementation of Stage 1.

Marks for your style will be allocated roughly according to the scheme below.

Style Marking Rubric

	0	1	2	3	4
Formatting (/5)					
Indentation (/2) - Should use a consistent indentation scheme.	Multiple instances throughout code of inconsistent/bad indentation	Code is mostly correctly indented	Code is consistently indented throughout the program		
Whitespace (/1) - Should use consistent whitespace (for example, 3 + 3 not 3+ 3)	Many whitespace errors	No whitespace errors			
Vertical Whitespace (/1) - Should use consistent whitespace (for example, vertical whitespace between sections of code)	Code has no consideration for use of vertical whitespace	Code consistently uses reasonable vertical whitespace			
Line Length (/1) - Lines should be max. 80 characters long	Many lines over 80 characters	No lines over 80 characters			
Documentation (/5)					

Comments (incl. header comment) (/3) - Comments have been used throughout the code above code sections and functions to explain their purpose. A header comment (with name, zID and a program description) has been included	No comments provided throughout code	Few comments provided throughout code	Comments are provided as needed, but some details or explanations may be missing causing the code to be difficult to follow	Comments have been used throughout the code above code sections and functions to explain their purpose. A header comment (with name, zID and a program description) has been included	
Function/variable/constant naming (/2) - Functions/variables/constants names all follow naming conventions in style guide and help in understanding the code	Functions/variables/constants names do not follow naming conventions in style guide and help in understanding the code	Functions/variables/constants names somewhat follow naming conventions in style guide and help in understanding the code	Functions/variables/constants names all follow naming conventions in style guide and help in understanding the code		
Organisation (/5)					
Function Usage (/4) - Code has been decomposed into appropriate functions separating functionalities	No functions are present, code is one main function	Some code has been moved to functions	Some code has been moved to sensible/thought out functions, and/or many functions exceed 50 lines (incl. main function)	Most code has been moved to sensible/thought out functions, and/or some functions exceed 50 lines (incl. main function)	All code h been meaningfu decompos into functions a maximu of 50 lines (incl. The main function)
Function Prototypes (/1) - Function Prototypes have been used to declare functions above main	Functions are used but have not been prototyped	All functions have a prototype above the main function or no functions are used			
Elegance (/5)					
Overdeep nesting (/2) - You should not have too many levels of nesting in your code (nesting which is 5 or more levels deep)	Many instances of overdeep nesting	<= 3 instances of overdeep nesting	No instances of overdeep nesting		
Code Repetition (/2) - Potential repetition of code has been dealt with via the use of functions or loops	Many instances of repeated code sections	<= 3 instances of repeated code sections	Potential repetition of code has been dealt with via the use of functions or loops		
Constant Usage (/1) - Any magic numbers are #defined	None of the constants used throughout program are #defined	All constants used are #defined and are used consistently in the code			
Illegal elements					
Illegal elements - Presence of any illegal elements indicated in the style guide	**CAP MARK AT 16/20**				

Note that the following penalties apply to your total mark for plagiarism:

0 for the assignment	Knowingly providing your work to anyone and it is subsequently submitted (by anyone).
0 for the assignment	Submitting any other person's work. This includes joint work.

0 FL for DPST1091	Paying another person to complete work. Submitting another person's work without their consent.

Allowed C Features

In this assignment, there are no restrictions on C Features, except for those in the style guide. If you choose to disregard this advice, you **must** still follow the style guide.

You also may be unable to get help from course staff if you use features not taught in DPST1091. Features that the Style Guide identifies as illegal will result in a penalty during marking. You can find the style marking rubric above. Please note that this assignment must be completed using only **Linked Lists** . Do not use arrays in this assignment. If you use arrays instead of lined lists you will receive a 0 for performance in this assignment.

Due Date

This assignment is due **Week 12 Friday 09:00am** (2025-08-01 09:00:00). For each day after that time, the maximum mark it can achieve will be reduced **by 5%** (off the ceiling). For example at:

- Less than 1 day (24 hours) past the due date, the maximum mark you can get is **95%**.
- Less than 2 days (48 hours) past the due date, the maximum mark you can get is **90%**.
- Less than 5 days (120 hours) past the due date, the maximum mark you can get is **75%**.

No submissions will be accepted at 5 days late, unless you have special provisions in place.

Change Log

Version 1.0
(2025-07-09 09:00)

- Assignment Released

DPST1091/CPTG1391 25T2: Programming Fundamentals!