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 3

In **Stage 3**, you'll be doing more advanced manipulation of 2D linked lists. You will also be managing your memory usage by freeing memory and preventing memory leaks. This milestone has been divided into four substages:

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

DANGER:

From this stage (**Stage 3**) onwards, **ALL** memory that you allocate in your program should be freed, and you should have no memory leaks.

You may have to go back and check your code from previous stages to make sure you don't accidentally `malloc` memory and never `free` it.

From now on, you should check for memory leaks by compiling with `--leak-check`. Autotests and marking tests from this stage onwards will check for memory leaks.

The full command you should be compiling with to ensure your code does not contain any leaks is `dcc cs_amusement_park.c main.c --leak-check -o cs_amusement_park`.

Stage 3.1 - End of Day

It is time for the park to start cleaning up and closing up for the day. This means closing all rides and having all visitors leave the park.

Command

```
q
```

Description

All malloc'd memory should be freed when the `q` command is used or when the program is ended with `CTRL+D`. This should also cause the program to print `Goodbye!` and end with no memory leaks.

Examples

– Example 3.1.1: Empty Park

Input:

```
UNSW
q
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c --leak-check -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: q

Goodbye!
```

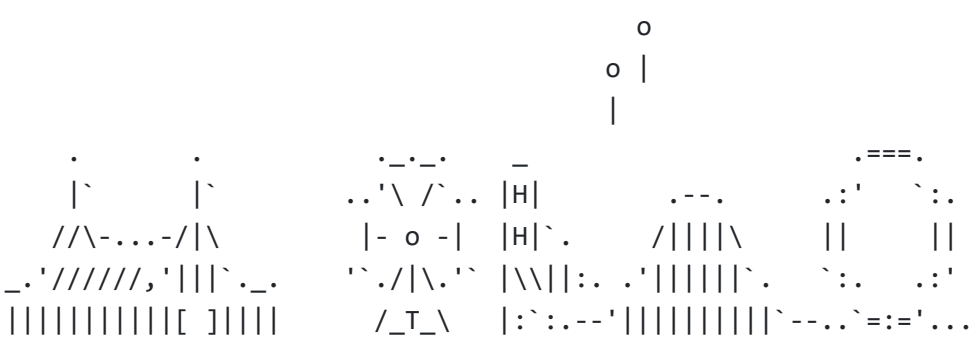
– Example 3.1.2: End of day with rides and visitors roaming

Input:

```
UNSW
a r RollerCoaster ROLLER_COASTER
a v Grace 160
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c --leak-check -o cs_amusement_park
$ ./cs_amusement_park
Welcome to CS Amusement Park!
```



```
Enter the name of the park: UNSW
Enter command: a r RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' added!
Enter command: a v Grace 160
Visitor: 'Grace' has entered the amusement park!
Enter command: [CTRL+D]
Goodbye!
```

- Example 3.1.3: End of day with rides and visitors in queues

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
p
q
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c --leak-check -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: p
==================[ UNSW ]==================
Rides:
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      Grace (165.00cm)
  MerryGoRound (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      Ibby (170.00cm)
Visitors:
  Sasha (160.50cm)
  Sofia (162.00cm)

Enter command: q

Goodbye!
```

Clarifications

- From stage 3.1 on, you should check for memory leaks by compiling with `--leak-check`, as seen in the examples below.
- Autotests and marking tests from this stage onwards will check for memory leaks.

> **NOTE:**
>
> You may like to autotest this section with the following command:
>
> ```
> 1091 autotest-stage 03_01 cs_amusement_park
> ```

Stage 3.2 - Leave the park

Command

```
l [visitor_name]
```

Description

The `l` command should attempt to find a visitor with a matching `visitor_name` and remove that visitor from the park, `free` ing all associated memory. Once completed the program should print `Visitor: '[visitor_name]' has left the park.`

Errors

- If no visitor containing `visitor_name` exists in the park or in any of the queues, then the following error should be printed: `ERROR: Visitor '[visitor_name]' not found in the park.` .

Examples

– Example 3.2.1: One visitor leaves

Input:

```
UNSW
a v Grace 160
l Grace
p
q
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c --leak-check -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 Grace 160
Visitor: 'Grace' has entered the amusement park!
Enter command: l Grace
Visitor: 'Grace' has left the park.
Enter command: p
==================[ UNSW ]==================
The amusement park is empty!

Enter command: q

Goodbye!
```

– Example 3.2.2: Multiple visitors in queues leave

Input:

```
UNSW
a r RollerCoaster ROLLER_COASTER
a v Sasha 160
a v Sofia 165
a v Ibby 170
j RollerCoaster Sofia
j RollerCoaster Sasha
j RollerCoaster Ibby
l Sasha
p
l Sofia
p
q
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c --leak-check -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 RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' added!
Enter command: a v Sasha 160
Visitor: 'Sasha' has entered the amusement park!
Enter command: a v Sofia 165
Visitor: 'Sofia' has entered the amusement park!
Enter command: a v Ibby 170
Visitor: 'Ibby' has entered the amusement park!
Enter command: j RollerCoaster Sofia
Visitor: 'Sofia' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Sasha
Visitor: 'Sasha' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Ibby
Visitor: 'Ibby' has entered the queue for 'RollerCoaster'.
Enter command: l Sasha
Visitor: 'Sasha' has left the park.
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Sofia (165.00cm)
      Ibby (170.00cm)
Visitors:
  No visitors!

Enter command: l Sofia
Visitor: 'Sofia' has left the park.
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Ibby (170.00cm)
Visitors:
  No visitors!

Enter command: q

Goodbye!
```

– Example 3.2.3: Errors with non-existent visitors

Input:

```
UNSW
a v Grace 160
l NonexistentVisitor
q
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c --leak-check -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 Grace 160
Visitor: 'Grace' has entered the amusement park!
Enter command: l NonexistentVisitor
ERROR: Visitor 'NonexistentVisitor' not found in the park.
Enter command: q

Goodbye!
```

Clarifications

- 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 visitor may be roaming the park or in the queue for a ride.

> **NOTE:**
>
> You may like to autotest this section with the following command:
>
> ```
> 1091 autotest-stage 03_02 cs_amusement_park
> ```

Stage 3.3 - Operate the rides

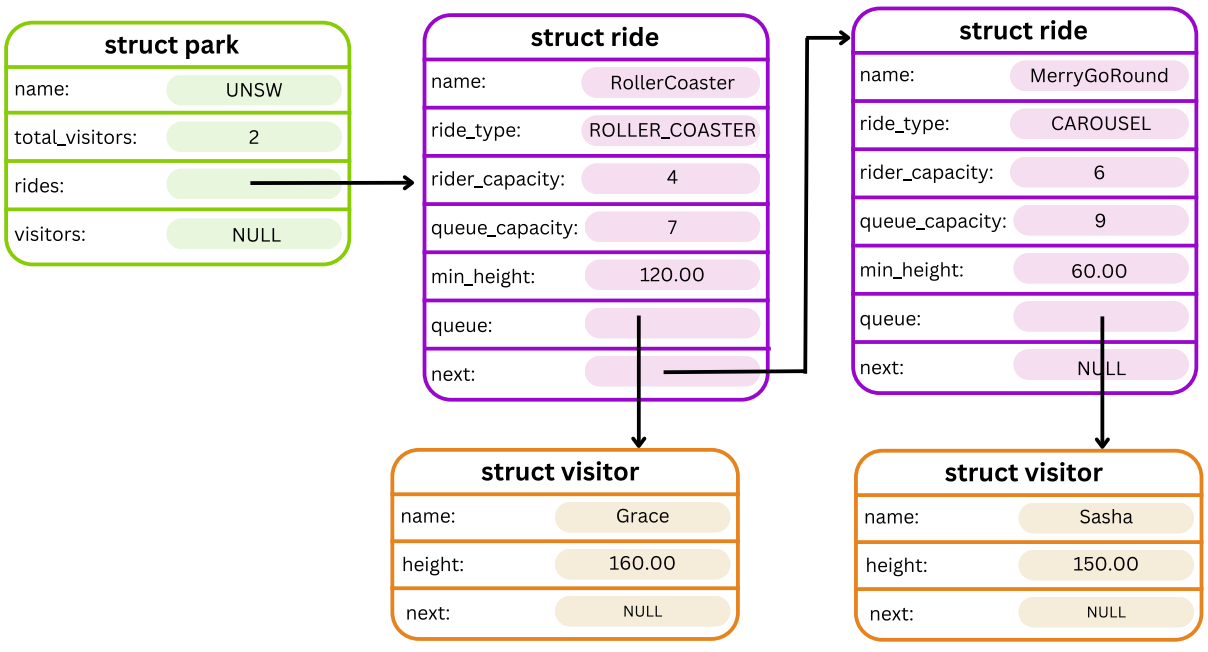
Command

```
r
```

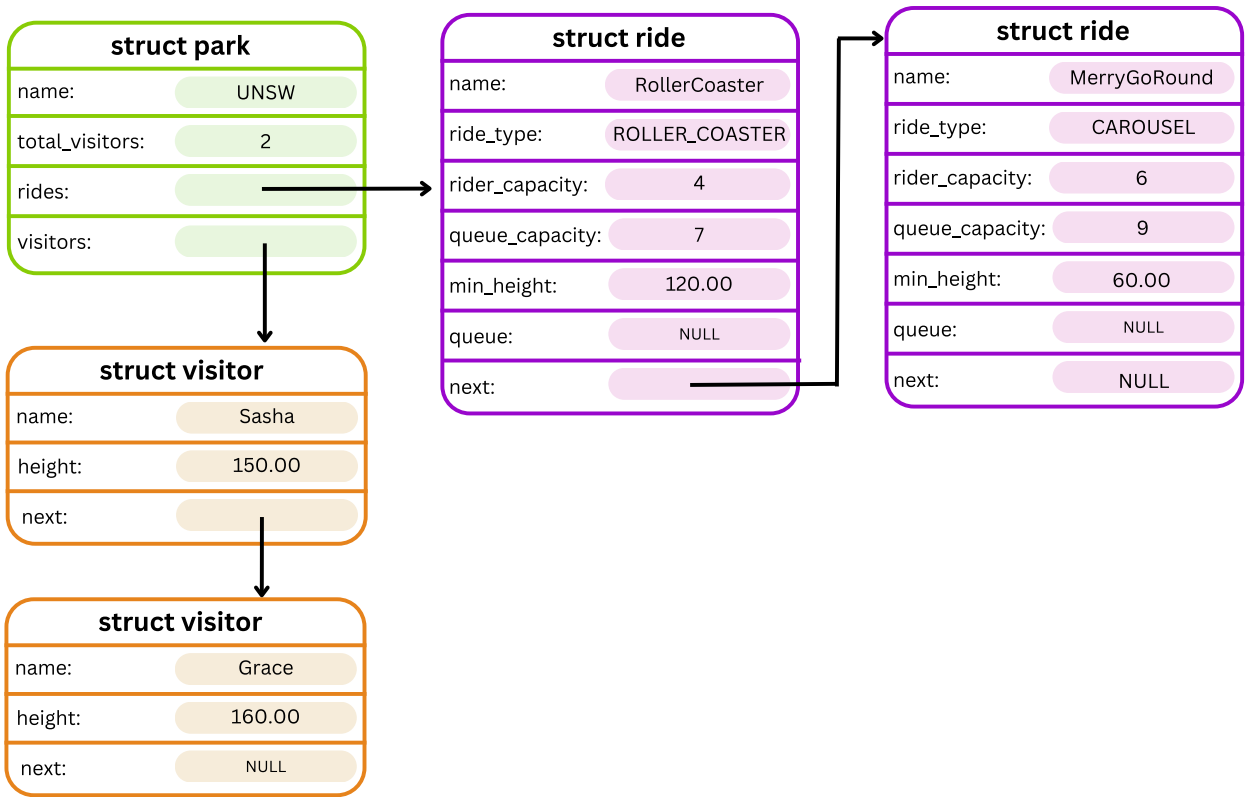
Description

The `r` command should operate all the rides such that each ride takes on the number of people equal to its rider capacity. Once the ride is complete, they exit the ride and are no longer in the ride's queue. Instead, they return to roaming the park as general park visitors. When these visitors return to roaming, they should be added to the end of the park's roaming list in the same order they were in the ride's queue. Additionally, rides should be processed in the reverse order of how they appear in the park list.

For example, before running the `r` command, the park will look something like this:



After running the command the park will look something like this:



Examples

- Example 3.3.1: Run rides simple

Input:

```
UNSW
a r RollerCoaster ROLLER_COASTER
a r Carousel CAROUSEL
a v Grace 160
a v Sasha 150
j RollerCoaster Grace
j Carousel Sasha
p
r
p
q
```

Input and Output:

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

Enter the name of the park: UNSW
Enter command: a r RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' added!
Enter command: a r Carousel CAROUSEL
Ride: 'Carousel' added!
Enter command: a v Grace 160
Visitor: 'Grace' has entered the amusement park!
Enter command: a v Sasha 150
Visitor: 'Sasha' has entered the amusement park!
Enter command: j RollerCoaster Grace
Visitor: 'Grace' has entered the queue for 'RollerCoaster'.
Enter command: j Carousel Sasha
Visitor: 'Sasha' has entered the queue for 'Carousel'.
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Grace (160.00cm)
  Carousel (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      Sasha (150.00cm)
Visitors:
  No visitors!

Enter command: r
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  Carousel (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      No visitors
Visitors:
  Sasha (150.00cm)
  Grace (160.00cm)

Enter command: q

Goodbye!
```

- Example 3.3.2: Single ride

Input:

```
UNSW
a r RollerCoaster ROLLER_COASTER
a v Grace 160
a v Sasha 150
a v Sofia 165
a v Ibby 175
a v Ben 172.5
a v Tammy 152
j RollerCoaster Grace
j RollerCoaster Sasha
j RollerCoaster Sofia
j RollerCoaster Tammy
j RollerCoaster Ben
j RollerCoaster Ibby
p
r
p
q
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c --leak-check -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 RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' added!
Enter command: a v Grace 160
Visitor: 'Grace' has entered the amusement park!
Enter command: a v Sasha 150
Visitor: 'Sasha' has entered the amusement park!
Enter command: a v Sofia 165
Visitor: 'Sofia' has entered the amusement park!
Enter command: a v Ibby 175
Visitor: 'Ibby' has entered the amusement park!
Enter command: a v Ben 172.5
Visitor: 'Ben' has entered the amusement park!
Enter command: a v Tammy 152
Visitor: 'Tammy' has entered the amusement park!
Enter command: j RollerCoaster Grace
Visitor: 'Grace' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Sasha
Visitor: 'Sasha' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Sofia
Visitor: 'Sofia' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Tammy
Visitor: 'Tammy' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Ben
Visitor: 'Ben' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Ibby
Visitor: 'Ibby' has entered the queue for 'RollerCoaster'.
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Grace (160.00cm)
      Sasha (150.00cm)
      Sofia (165.00cm)
      Tammy (152.00cm)
      Ben (172.50cm)
      Ibby (175.00cm)
Visitors:
  No visitors!

Enter command: r
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Ben (172.50cm)
      Ibby (175.00cm)
Visitors:
  Grace (160.00cm)
  Sasha (150.00cm)
  Sofia (165.00cm)
```

```
  Tammy (152.00cm)

Enter command: q

Goodbye!
```

– Example 3.3.3: Multiple rides

Input:

```
UNSW
a r RollerCoaster ROLLER_COASTER
a v Grace 160
a v Sasha 150
a v Sofia 165
i 1 Carousel CAROUSEL
a v Ibby 175
a v Ben 172.5
a v Tammy 152
j RollerCoaster Grace
j Carousel Sasha
j Carousel Sofia
a v Andrew 180
j Carousel Andrew
a v Angela 164
j RollerCoaster Tammy
j Carousel Ben
j RollerCoaster Ibby
j RollerCoaster Angela
p
r
p
q
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c --leak-check -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 RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' added!
Enter command: a v Grace 160
Visitor: 'Grace' has entered the amusement park!
Enter command: a v Sasha 150
Visitor: 'Sasha' has entered the amusement park!
Enter command: a v Sofia 165
Visitor: 'Sofia' has entered the amusement park!
Enter command: i 1 Carousel CAROUSEL
Ride: 'Carousel' inserted!
Enter command: a v Ibby 175
Visitor: 'Ibby' has entered the amusement park!
Enter command: a v Ben 172.5
Visitor: 'Ben' has entered the amusement park!
Enter command: a v Tammy 152
Visitor: 'Tammy' has entered the amusement park!
Enter command: j RollerCoaster Grace
Visitor: 'Grace' has entered the queue for 'RollerCoaster'.
Enter command: j Carousel Sasha
Visitor: 'Sasha' has entered the queue for 'Carousel'.
Enter command: j Carousel Sofia
Visitor: 'Sofia' has entered the queue for 'Carousel'.
Enter command: a v Andrew 180
Visitor: 'Andrew' has entered the amusement park!
Enter command: j Carousel Andrew
Visitor: 'Andrew' has entered the queue for 'Carousel'.
Enter command: a v Angela 164
Visitor: 'Angela' has entered the amusement park!
Enter command: j RollerCoaster Tammy
Visitor: 'Tammy' has entered the queue for 'RollerCoaster'.
Enter command: j Carousel Ben
Visitor: 'Ben' has entered the queue for 'Carousel'.
Enter command: j RollerCoaster Ibby
Visitor: 'Ibby' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Angela
Visitor: 'Angela' has entered the queue for 'RollerCoaster'.
Enter command: p
==================[ UNSW ]==================
Rides:
  Carousel (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      Sasha (150.00cm)
      Sofia (165.00cm)
      Andrew (180.00cm)
      Ben (172.50cm)
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Grace (160.00cm)
      Tammy (152.00cm)
      Ibby (175.00cm)
      Angela (164.00cm)
Visitors:
```

```
  No visitors!

Enter command: r
Enter command: p
==================[ UNSW ]==================
Rides:
  Carousel (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      No visitors
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
Visitors:
  Grace (160.00cm)
  Tammy (152.00cm)
  Ibby (175.00cm)
  Angela (164.00cm)
  Sasha (150.00cm)
  Sofia (165.00cm)
  Andrew (180.00cm)
  Ben (172.50cm)

Enter command: q

Goodbye!
```

Clarifications

- Rides are processed in reverse order of their appearance in the linked list, and visitors from each ride are appended to the roaming list in the same order as they appeared in the ride's queue.
- If there are no rides in the park, the `r` command should do nothing.
- If a ride's queue is empty, no visitors are processed for that ride, and the ride is simply skipped.

> **NOTE:**
>
> You may like to autotest this section with the following command:
>
> ```
> 1091 autotest-stage 03_03 cs_amusement_park
> ```

Stage 3.4 - Ride Shut Down

Command

```
S [ride_name]
```

Description

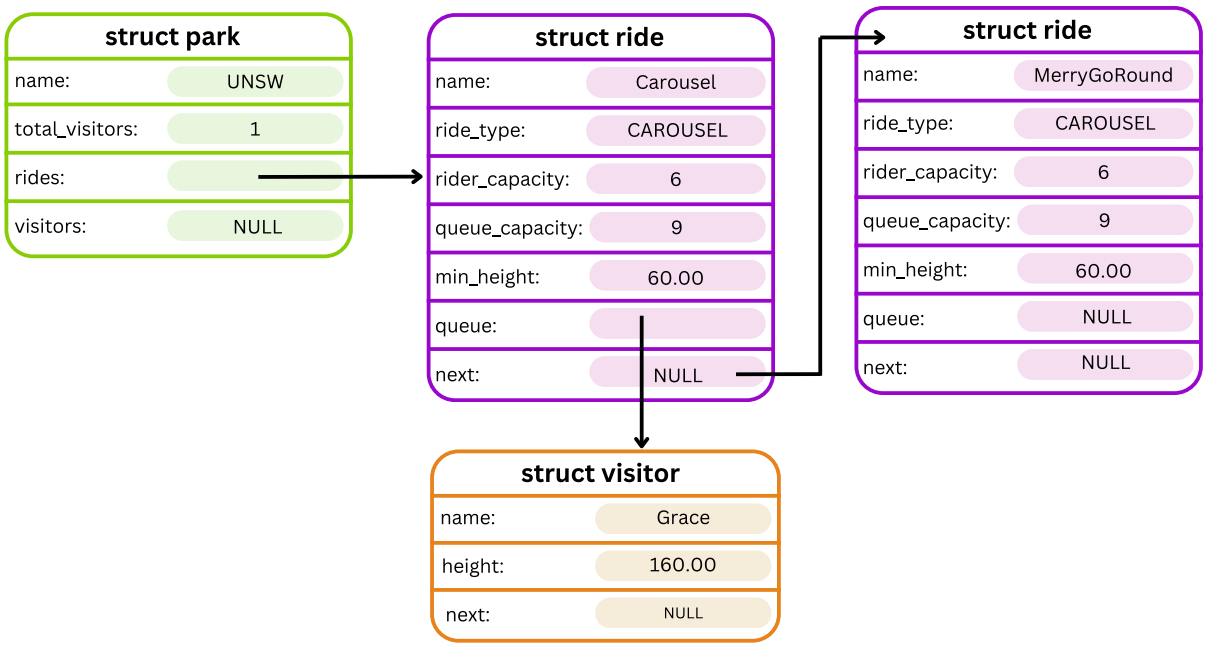
Oh no! A ride has malfunctioned and needs to shut down for the day! The `S` command should shut down the ride with the name `ride_name` and move all of the visitors in the queue for that ride to the next ride(s) of the same type.

The visitors in the queue of the shutting-down ride are first added to the queue of the first available ride of the same type, filling it to its capacity.

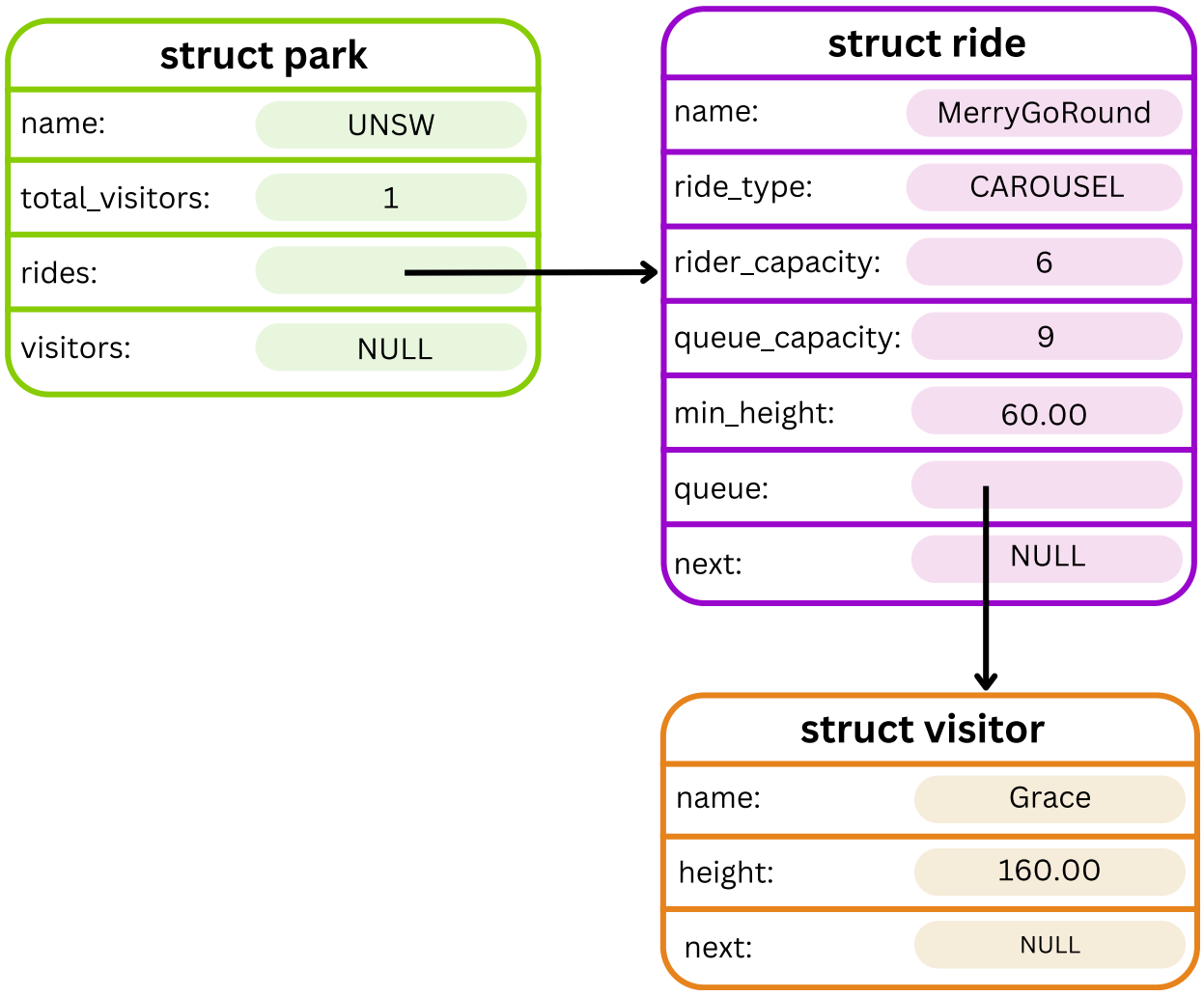
- If there are still remaining visitors after filling the first ride, they are then transferred to the next available ride of the same type, and so on, until all visitors are moved.
- If the total capacity of the available rides of the same type is insufficient to accommodate all the visitors from the malfunctioning ride, then none of the visitors will be added to any ride queue. Instead, all visitors from the queue of the malfunctioning ride will be added back to the list of visitors roaming the park.

After handling the visitors, the malfunctioning ride must be removed from the park, and its memory freed.

For example, before a ride shutdown, the park may look something like this:



After running the command `S Carousel` the park will look something like this:



Errors

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

  `ERROR: No ride exists with name: '[ride_name]'.`

- If there is not enough space to redistribute all visitors the following error should dbe printer:

  `ERROR: Not enough capacity to redistribute all visitors from '[ride_name]'.`

Examples

– Example 3.4.1: Simple ride shutdown

Input:

```
UNSW
a r RollerCoaster ROLLER_COASTER
a r Carousel CAROUSEL
a r SwingSet CAROUSEL
a v Grace 160
j Carousel Grace
S Carousel
p
q
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c --leak-check -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 RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' added!
Enter command: a r Carousel CAROUSEL
Ride: 'Carousel' added!
Enter command: a r SwingSet CAROUSEL
Ride: 'SwingSet' added!
Enter command: a v Grace 160
Visitor: 'Grace' has entered the amusement park!
Enter command: j Carousel Grace
Visitor: 'Grace' has entered the queue for 'Carousel'.
Enter command: S Carousel
Ride: 'Carousel' shut down.
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  SwingSet (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      Grace (160.00cm)
Visitors:
  No visitors!

Enter command: q

Goodbye!
```

- Example 3.4.2: Shut down ride with insufficient rides of the same type

Input:

```
UNSW
a r RollerCoaster ROLLER_COASTER
a r Carousel CAROUSEL
a v Grace 160
a v Sasha 150
j Carousel Grace
j Carousel Sasha
S Carousel
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c --leak-check -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 RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' added!
Enter command: a r Carousel CAROUSEL
Ride: 'Carousel' added!
Enter command: a v Grace 160
Visitor: 'Grace' has entered the amusement park!
Enter command: a v Sasha 150
Visitor: 'Sasha' has entered the amusement park!
Enter command: j Carousel Grace
Visitor: 'Grace' has entered the queue for 'Carousel'.
Enter command: j Carousel Sasha
Visitor: 'Sasha' has entered the queue for 'Carousel'.
Enter command: S Carousel
ERROR: Not enough capacity to redistribute all visitors from 'Carousel'.
Ride: 'Carousel' shut down.
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
Visitors:
  Grace (160.00cm)
  Sasha (150.00cm)

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

- Example 3.4.2: Complex multi ride shutdown

Input:

```
UNSW
a r RollerCoaster ROLLER_COASTER
a r LoopyRoller ROLLER_COASTER
a r MegaRoller ROLLER_COASTER
a v Sasha 160
a v Tammy 170
a v Sofia 165
a v Grace 155
a v Ibby 150
a v Ben 145
a v Andrew 175
j RollerCoaster Sofia
j RollerCoaster Andrew
j RollerCoaster Sasha
j RollerCoaster Ben
j RollerCoaster Tammy
j RollerCoaster Ibby
j RollerCoaster Grace
p
a v Angela 164
j LoopyRoller Angela
S RollerCoaster
p
S LoopyRoller
p
[CTRL+D]
```

Input and Output:

```
$ dcc cs_amusement_park.c main.c --leak-check -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 RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' added!
Enter command: a r LoopyRoller ROLLER_COASTER
Ride: 'LoopyRoller' added!
Enter command: a r MegaRoller ROLLER_COASTER
Ride: 'MegaRoller' added!
Enter command: a v Sasha 160
Visitor: 'Sasha' has entered the amusement park!
Enter command: a v Tammy 170
Visitor: 'Tammy' has entered the amusement park!
Enter command: a v Sofia 165
Visitor: 'Sofia' has entered the amusement park!
Enter command: a v Grace 155
Visitor: 'Grace' has entered the amusement park!
Enter command: a v Ibby 150
Visitor: 'Ibby' has entered the amusement park!
Enter command: a v Ben 145
Visitor: 'Ben' has entered the amusement park!
Enter command: a v Andrew 175
Visitor: 'Andrew' has entered the amusement park!
Enter command: j RollerCoaster Sofia
Visitor: 'Sofia' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Andrew
Visitor: 'Andrew' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Sasha
Visitor: 'Sasha' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Ben
Visitor: 'Ben' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Tammy
Visitor: 'Tammy' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Ibby
Visitor: 'Ibby' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Grace
Visitor: 'Grace' has entered the queue for 'RollerCoaster'.
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Sofia (165.00cm)
      Andrew (175.00cm)
      Sasha (160.00cm)
      Ben (145.00cm)
      Tammy (170.00cm)
      Ibby (150.00cm)
      Grace (155.00cm)
  LoopyRoller (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  MegaRoller (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
```

```
    Minimum Height: 120.00cm
    Queue:
      No visitors
Visitors:
  No visitors!

Enter command: a v Angela 164
Visitor: 'Angela' has entered the amusement park!
Enter command: j LoopyRoller Angela
Visitor: 'Angela' has entered the queue for 'LoopyRoller'.
Enter command: S RollerCoaster
Ride: 'RollerCoaster' shut down.
Enter command: p
==================[ UNSW ]==================
Rides:
  LoopyRoller (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Angela (164.00cm)
      Sofia (165.00cm)
      Andrew (175.00cm)
      Sasha (160.00cm)
      Ben (145.00cm)
      Tammy (170.00cm)
      Ibby (150.00cm)
  MegaRoller (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Grace (155.00cm)
Visitors:
  No visitors!

Enter command: S LoopyRoller
ERROR: Not enough capacity to redistribute all visitors from 'LoopyRoller'.
Ride: 'LoopyRoller' shut down.
Enter command: p
==================[ UNSW ]==================
Rides:
  MegaRoller (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Grace (155.00cm)
Visitors:
  Angela (164.00cm)
  Sofia (165.00cm)
  Andrew (175.00cm)
  Sasha (160.00cm)
  Ben (145.00cm)
  Tammy (170.00cm)
  Ibby (150.00cm)

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

Clarifications

- After the visitors from the malfunctioning ride have been successfully transferred, that ride must be removed from the park and its memory `free` 'd.
- If a ride's queue is already at full capacity, no additional visitors should be added to it.
- The available rides of the same type should be processed in the order they appear in the park.
- Visitors are moved to the first ride until it reaches capacity, then overflow to the next, and so on.
- If there aren't enough rides of the same type to accommodate all visitors affected by a ride shut down, none will be added to any ride queues. Instead, they'll return to the roaming visitor list in their original order.

NOTE:

You may like to autotest this section with the following command:

```
1091 autotest-stage 03_04 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 03 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!