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 4

This stage is for students who want to challenge themselves, and solve more complicated linked lists and programming problems. This milestone has been divided into three substages:

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

Stage 4.1 - Merge

Command

```
M [ride_type]
```

Description

Merge the two rides of the given `ride_type` that have the smallest queue lengths into a single larger ride, combining their queues and capacities. The visitor queues from both rides should be merged alternately into the smaller ride, starting with the queue of the ride that has the longer queue, while the other ride is removed. If the queues are the same length, start with the ride that appears first, closest to the front of the park, in the list of rides. In this case the ride whose closer to the head of the park remains in the park, while the other ride is removed.

On success the program should print `Merged the two smallest rides of type '[ride_type]'.` .

Errors

- If there is not enough rides, i.e. 2 or more rides of the same type in the park, the following error should be printed:
  `ERROR: Not enough rides of the specified type to merge.`

Examples

– Example 4.1.1: Simple merge

Input:

```
UNSW
a r RollerCoaster_1 ROLLER_COASTER
a r RollerCoaster_2 ROLLER_COASTER
a v Sasha 150
a v Sofia 160
a v Grace 155
a v Ibby 165
j RollerCoaster_1 Sasha
j RollerCoaster_1 Sofia
j RollerCoaster_2 Grace
j RollerCoaster_2 Ibby
p
M ROLLER_COASTER
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_1 ROLLER_COASTER
Ride: 'RollerCoaster_1' added!
Enter command: a r RollerCoaster_2 ROLLER_COASTER
Ride: 'RollerCoaster_2' added!
Enter command: a v Sasha 150
Visitor: 'Sasha' has entered the amusement park!
Enter command: a v Sofia 160
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 165
Visitor: 'Ibby' has entered the amusement park!
Enter command: j RollerCoaster_1 Sasha
Visitor: 'Sasha' has entered the queue for 'RollerCoaster_1'.
Enter command: j RollerCoaster_1 Sofia
Visitor: 'Sofia' has entered the queue for 'RollerCoaster_1'.
Enter command: j RollerCoaster_2 Grace
Visitor: 'Grace' has entered the queue for 'RollerCoaster_2'.
Enter command: j RollerCoaster_2 Ibby
Visitor: 'Ibby' has entered the queue for 'RollerCoaster_2'.
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster_1 (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Sasha (150.00cm)
      Sofia (160.00cm)
  RollerCoaster_2 (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Grace (155.00cm)
      Ibby (165.00cm)
Visitors:
  No visitors!

Enter command: M ROLLER_COASTER
Merged the two smallest rides of type 'ROLLER_COASTER'.
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster_1 (ROLLER_COASTER)
    Rider Capacity: 8
    Queue Capacity: 14
    Minimum Height: 120.00cm
    Queue:
      Sasha (150.00cm)
      Grace (155.00cm)
      Sofia (160.00cm)
      Ibby (165.00cm)
Visitors:
  No visitors!

Enter command: q
```

```
Goodbye!
```

– Example 4.1.2: Not enough rides

Input:

```
UNSW
a r Carousel_1 CAROUSEL
p
M 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 Carousel_1 CAROUSEL
Ride: 'Carousel_1' added!
Enter command: p
==================[ UNSW ]==================
Rides:
  Carousel_1 (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      No visitors
Visitors:
  No visitors!

Enter command: M CAROUSEL
ERROR: Not enough rides of the specified type to merge.
Enter command: p
==================[ UNSW ]==================
Rides:
  Carousel_1 (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      No visitors
Visitors:
  No visitors!

Enter command: q

Goodbye!
```

– Example 4.1.3: Complex merge

Input:

```
UNSW
a r FerrisWheel_1 FERRIS_WHEEL
a r FerrisWheel_2 FERRIS_WHEEL
a v Sofia 155
a v Grace 165
a v Ibby 170
j FerrisWheel_1 Grace
j FerrisWheel_1 Sofia
j FerrisWheel_2 Ibby
p
M FERRIS_WHEEL
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 FerrisWheel_1 FERRIS_WHEEL
Ride: 'FerrisWheel_1' added!
Enter command: a r FerrisWheel_2 FERRIS_WHEEL
Ride: 'FerrisWheel_2' added!
Enter command: a v Sofia 155
Visitor: 'Sofia' has entered the amusement park!
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 FerrisWheel_1 Grace
Visitor: 'Grace' has entered the queue for 'FerrisWheel_1'.
Enter command: j FerrisWheel_1 Sofia
Visitor: 'Sofia' has entered the queue for 'FerrisWheel_1'.
Enter command: j FerrisWheel_2 Ibby
Visitor: 'Ibby' has entered the queue for 'FerrisWheel_2'.
Enter command: p
==================[ UNSW ]==================
Rides:
  FerrisWheel_1 (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      Grace (165.00cm)
      Sofia (155.00cm)
  FerrisWheel_2 (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      Ibby (170.00cm)
Visitors:
  No visitors!

Enter command: M FERRIS_WHEEL
Merged the two smallest rides of type 'FERRIS_WHEEL'.
Enter command: p
==================[ UNSW ]==================
Rides:
  FerrisWheel_2 (FERRIS_WHEEL)
    Rider Capacity: 16
    Queue Capacity: 22
    Minimum Height: 75.00cm
    Queue:
      Grace (165.00cm)
      Ibby (170.00cm)
      Sofia (155.00cm)
Visitors:
  No visitors!

Enter command: q

Goodbye!
```

Clarifications

- If there are no two rides of the `type`, then the `M` command should do nothing.
- Any remaining visitors from the longer queue should be appended to the end of the merged queue, in the same order in which they were in the original queue.
- The visitor queues from both rides should be merged alternately into the smaller ride, starting with the queue of the ride that has the longer queue.
- If the queues are the same length, the first ride in the list should start the combined queue and be preserved.
- The merged queue should be assigned to the ride that appears first in the list of rides in the park (i.e., the one closer to the head of the list).

> **NOTE:**
>
> You may like to autotest this section with the following command:
>
> ```
> 1091 autotest-stage 04_01 cs_amusement_park
> ```

Stage 4.2 - Split

Command

```
s [n] [ride_name]
```

Description

The `s` command enables splitting an existing ride into multiple smaller rides, each with a portion of the original queue of visitors. The command takes two arguments:

1. `n`: The number of new rides to create.
2. `ride_name`: The name of the original ride to be split.

It should then replace the ride with the name `ride_name`, in place, with `n` new rides. These new rides should be named sequentially as `name_1, name_2, ..., name_n` with the suffix `_i`. The queue of visitors originally in the queue of the ride being split is then divided among the `n` new rides. The division should aim to distribute the visitors as evenly as possible, ensuring a fair allocation.

- If the number of visitors does not divide evenly, some rides may have one more visitor than others, starting from the first of the new split rides in sequential order.
- If the original ride name is too long to accommodate the `_i` suffix within the maximum allowed length for ride names, it should be truncated to make space for the suffix.

On success the program should print `Ride '[ride_name]' split into [n] new rides.`.

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 `n <= 1`, then the following error should be printed:

  `ERROR: Cannot split '[ride_name]' into [n] rides. n must be > 1.`

Examples

– Example 4.2.1: Simple split

Input:

```
UNSW
a r RollerCoaster ROLLER_COASTER
a v Sasha 150
a v Grace 160
a v Sofia 155
a v Andrew 165
a v Ibby 170
j RollerCoaster Sasha
j RollerCoaster Sofia
j RollerCoaster Andrew
j RollerCoaster Grace
j RollerCoaster Ibby
p
s 3 RollerCoaster
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 150
Visitor: 'Sasha' has entered the amusement park!
Enter command: a v Grace 160
Visitor: 'Grace' has entered the amusement park!
Enter command: a v Sofia 155
Visitor: 'Sofia' has entered the amusement park!
Enter command: a v Andrew 165
Visitor: 'Andrew' has entered the amusement park!
Enter command: a v Ibby 170
Visitor: 'Ibby' has entered the amusement park!
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 Andrew
Visitor: 'Andrew' has entered the queue for 'RollerCoaster'.
Enter command: j RollerCoaster Grace
Visitor: 'Grace' 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:
      Sasha (150.00cm)
      Sofia (155.00cm)
      Andrew (165.00cm)
      Grace (160.00cm)
      Ibby (170.00cm)
Visitors:
  No visitors!

Enter command: s 3 RollerCoaster
Ride 'RollerCoaster' split into 3 new rides.
Enter command: p
==================[ UNSW ]==================
Rides:
  RollerCoaster_1 (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Sasha (150.00cm)
      Sofia (155.00cm)
  RollerCoaster_2 (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Andrew (165.00cm)
      Grace (160.00cm)
  RollerCoaster_3 (ROLLER_COASTER)
```

```
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Ibby (170.00cm)
Visitors:
  No visitors!

Enter command: q

Goodbye!
```

- Example 4.2.2: Complex split

Input:

```
UNSW
a r test_1 ROLLER_COASTER
a r test_3 ROLLER_COASTER
a r test ROLLER_COASTER
a v Sasha 150
a v Andrew 160
a v Tammy 155
a v Ben 165
j test Sasha
j test Andrew
j test Tammy
j test Ben
p
s 3 test
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 test_1 ROLLER_COASTER
Ride: 'test_1' added!
Enter command: a r test_3 ROLLER_COASTER
Ride: 'test_3' added!
Enter command: a r test ROLLER_COASTER
Ride: 'test' added!
Enter command: a v Sasha 150
Visitor: 'Sasha' has entered the amusement park!
Enter command: a v Andrew 160
Visitor: 'Andrew' has entered the amusement park!
Enter command: a v Tammy 155
Visitor: 'Tammy' has entered the amusement park!
Enter command: a v Ben 165
Visitor: 'Ben' has entered the amusement park!
Enter command: j test Sasha
Visitor: 'Sasha' has entered the queue for 'test'.
Enter command: j test Andrew
Visitor: 'Andrew' has entered the queue for 'test'.
Enter command: j test Tammy
Visitor: 'Tammy' has entered the queue for 'test'.
Enter command: j test Ben
Visitor: 'Ben' has entered the queue for 'test'.
Enter command: p
==================[ UNSW ]==================
Rides:
  test_1 (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  test_3 (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  test (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      Sasha (150.00cm)
      Andrew (160.00cm)
      Tammy (155.00cm)
      Ben (165.00cm)
Visitors:
  No visitors!

Enter command: s 3 test
Ride 'test' split into 3 new rides.
Enter command: p
==================[ UNSW ]==================
Rides:
  test_1 (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
```

```
  Queue:
    No visitors
test_3 (ROLLER_COASTER)
  Rider Capacity: 4
  Queue Capacity: 7
  Minimum Height: 120.00cm
  Queue:
    No visitors
test_2 (ROLLER_COASTER)
  Rider Capacity: 4
  Queue Capacity: 7
  Minimum Height: 120.00cm
  Queue:
    Sasha (150.00cm)
    Andrew (160.00cm)
test_4 (ROLLER_COASTER)
  Rider Capacity: 4
  Queue Capacity: 7
  Minimum Height: 120.00cm
  Queue:
    Tammy (155.00cm)
test_5 (ROLLER_COASTER)
  Rider Capacity: 4
  Queue Capacity: 7
  Minimum Height: 120.00cm
  Queue:
    Ben (165.00cm)
Visitors:
  No visitors!

Enter command: q

Goodbye!
```

- Example 4.2.3: Overly long split

Input:

```
UNSW
a r a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_len
ROLLER_COASTER
p
s 3 a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_len
p
s 5 a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_2
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
a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_len
ROLLER_COASTER
Ride: 'a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_len'
added!
Enter command: p
==================[ UNSW ]==================
Rides:
  a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_len
(ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
Visitors:
  No visitors!

Enter command: s 3
a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_len
Ride 'a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_len' split
into 3 new rides.
Enter command: p
==================[ UNSW ]==================
Rides:
  a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_1
(ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_2
(ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_3
(ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
Visitors:
  No visitors!

Enter command: s 5
a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_2
Ride 'a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_2' split
into 5 new rides.
Enter command: p
==================[ UNSW ]==================
Rides:
  a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_1
(ROLLER_COASTER)
```

```
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_4
(ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_5
(ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_6
(ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_7
(ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_8
(ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  a_very_long_string_that_will_need_to_be_sliced_when_running_the_split_command_to_fit_in_the_max_l_3
(ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
Visitors:
  No visitors!

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

Clarifications

- The division should aim to distribute the visitors as evenly as possible.
- If the original ride name is too long to accommodate the `_i` suffix within the maximum allowed length for ride names, it should be truncated to make space for the suffix.
- If some of the sequentially numbered ride names already exist in the park, the new rides should be given the next available sequential names that do not conflict with existing ride names.
- The upperbound of `n` is the maximum size of an integer in c.
- If the number of visitors does not divide evenly, some rides may have one more visitor than others, starting from the head of the list of rides in the park.

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

```
1091 autotest-stage 04_02 cs_amusement_park
```

Stage 4.3 - Scheduled

For this sub-stage you will be implementing two commands:

- `T [n] [command]`
- `~ [n]`

Command 1

```
T [n] [command]
```

Description

The `T` command schedules a command to run `n` ticks into the future. Once scheduled, it will execute automatically when the simulation advances by `n` number of ticks. The command takes two arguments:

1. `n` : The number of ticks from the current moment after which the command should execute.
2. `[command]` : Any valid command that could normally be issued in the program.

Scheduling with `T` is instantaneous and does not advance the simulation. Scheduled commands are stored and tracked until their countdown reaches zero and only then should they execute.

At each tick, all scheduled commands whose countdowns reach zero are executed before processing any manual commands entered on that tick. If multiple commands are scheduled for the same tick, they execute in the order they were scheduled.

Note, **time in the simulation advances by one tick when any command other than `T` and `~` , such as `a v Sofia 160` is entered.**

On success, the program should print `Command scheduled to run in [n] ticks.` .

Command 2

```
~ [n]
```

Description

The `~` command moves the schedule forward by a single unit of time, also referred to as a tick. The command takes one argument `n` which is the number of ticks the program should progress by.

The `~` command itself does not progress time by 1 tick, like entering a command such as `a v Sofia 160` would. Instead it progresses time by `n` numbers of ticks.

> **NOTE:**
>
> For this stage, you are **not permitted to use arrays** to store scheduled commands, visitors, or rides. You must manage scheduled commands using a **linked list**.

Errors

- If `n < 1` , then the following error should be printed:
  `ERROR: Invalid tick delay: [n]. Must be > 0.`
- If the quit command `q` is scheduled, the following error should printed only when it is executed, not when scheduled:
  `ERROR: Cannot schedule quit command.`
- If the scheduled `[command]` is invalid, the error is printed only when it is executed, not when scheduled.

Examples

– Example 4.3.1: Simple Scheduled

Input:

```
UNSW
T 2 a v Grace 150
a v Sofia 160
a v Sasha 160
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: T 2 a v Grace 150
Command scheduled to run in 2 ticks.
Enter command: a v Sofia 160
Visitor: 'Sofia' has entered the amusement park!
Enter command: a v Sasha 160
Visitor: 'Grace' has entered the amusement park!
Visitor: 'Sasha' has entered the amusement park!
Enter command: p
==================[ UNSW ]==================
Rides:
  No rides!
Visitors:
  Sofia (160.00cm)
  Grace (150.00cm)
  Sasha (160.00cm)

Enter command: q

Goodbye!
```

– Example 4.3.2: Complex Scheduled

Input:

```
UNSW
a r BigWheel FERRIS_WHEEL
T 2 s 2 BigWheel
~ 2
p
T 5 M FERRIS_WHEEL
a r One FERRIS_WHEEL
a r Two FERRIS_WHEEL
~ 3
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: T 2 s 2 BigWheel
Command scheduled to run in 2 ticks.
Enter command: ~ 2
Ride 'BigWheel' split into 2 new rides.
Enter command: p
==================[ UNSW ]==================
Rides:
  BigWheel_1 (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
  BigWheel_2 (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
Visitors:
  No visitors!

Enter command: T 5 M FERRIS_WHEEL
Command scheduled to run in 5 ticks.
Enter command: a r One FERRIS_WHEEL
Ride: 'One' added!
Enter command: a r Two FERRIS_WHEEL
Ride: 'Two' added!
Enter command: ~ 3
Merged the two smallest rides of type 'FERRIS_WHEEL'.
Enter command: p
==================[ UNSW ]==================
Rides:
  BigWheel_1 (FERRIS_WHEEL)
    Rider Capacity: 16
    Queue Capacity: 22
    Minimum Height: 75.00cm
    Queue:
      No visitors
  One (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
  Two (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
Visitors:
  No visitors!

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

Clarifications

- `n` is the number of ticks from the current moment after which the command should be executed.
- The upperbound of `n` is the maximum size of an integer in c.
- `[command]` is any valid command that would normally be issued at that moment in time.
- Scheduled commands are automatically executed once their tick countdown reaches zero.
- Commands scheduled for the same tick are resolved in the order they were scheduled.
- `T` itself does not consume a tick. Scheduling is instantaneous and does not advance the simulation. However, any other command or an explicit tick-advance action, the `~` command moves the simulation forward.
- Scheduled commands resolve before processing any "manual" command issued on that tick.
- You may assume that you cannot schedule the `T` or `~` commands, and these cases do not need to be handled, nor will be tested.

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 04 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!