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"
        "                                         |\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 1

For Stage 1 of this assignment, you will be implementing some basic commands to set up your amusement park simulator! This milestone has been divided into five substages:

- 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 1.1 - Creating the park

As you've probably found out by now, it can be really handy to have a function that `malloc` s, initialises, and returns a single linked list node.

So in **Stage 1.1**, we will be implementing a function which does exactly that for a `struct park` , `struct ride` and `struct visitor` .

You'll find the following unimplemented function stubs in `cs_amusement_park.c` , in the starter code:

– Function Stubs

```
// Stage 1.1
// Function to initialise the park
// Params:
//      name - the name of the park
// Returns: a pointer to the park
struct park *initialise_park(char name[MAX_SIZE]) {
    // TODO: Replace this with your code
    printf("initialise park not yet implemented\n");
    return NULL;
}
```

```
// Stage 1.1
// Function to create a visitor
// Params:
//      name - the name of the visitor
//      height - the height of the visitor
// Returns: a pointer to the visitor
struct visitor *create_visitor(char name[MAX_SIZE], double height) {
    // TODO: Replace this with your code
    printf("Create visitor not yet implemented\n");
    return NULL;
}
```

```
// Stage 1.1
// Function to create a ride
// Params:
//      name - the name of the ride
//      type - the type of the ride
// Returns: a pointer to the ride
struct ride *create_ride(char name[MAX_SIZE], enum ride_type type) {
    // TODO: Replace this with your code
    printf("Create ride not yet implemented\n");
    return NULL;
}
```

NOTE:

We would call this a function "stub", because it's just the basic structure of the functions, but without any real code inside.

You'll be writing code inside this function stub to make it work :)

Your task is to complete these functions, so that they:

1. Malloc the space required.
2. Copy in all the initial data.
3. Initialises all other fields to some reasonable value.
4. Return the relevant struct.

HINT:

`name` is an array of chars (a string), so `new_ride->name = name` won't work!

You may need to look up the function `strcpy` .

When creating rides, the capacity and minimum height will be added based on the enums.

Type	Rider Capacity	Queue Capacity	Min Height (cm)
`ROLLER_COASTER`	4	7	120
`CAROUSEL`	6	9	60
`FERRIS_WHEEL`	8	11	75
`BUMPER_CARS`	10	13	100

Clarifications

- No error handling is required for this stage.
- The initialised park should contain a total of 0 visitors and 0 rides to begin with.

Testing

There are no autotests for **Stage 1.1**.

Instead, double-check your work by compiling your code using `dcc` and ensure there are no warnings or errors.

You could also write some temporary testing code to check that your functions work properly when they run.

For example, you could copy the following testing code *into your main function* in `main.c` :

– Main function for testing

```c
#include <string.h>
#include <stdio.h>

#include "cs_amusement_park.h"

int main(void) {
    // Initialise the park
    char park_name[MAX_SIZE];
    strcpy(park_name, "UNSW");

    struct park *my_park = initialise_park(park_name);

    // Print park details
    printf("Park initialised:\n");
    printf("  Name: %s\n", my_park->name);
    printf("  Total Visitors: %d\n", my_park->total_visitors);
    if (my_park->rides == NULL) {
        printf("  Rides: NULL\n");
    } else {
        printf("  Rides not set to NULL\n");
    }
    if (my_park->visitors == NULL) {
        printf("  Visitors: NULL\n\n");
    } else {
        printf("  Visitors not set to NULL\n\n");
    }

    // Create a ride
    char ride_name[MAX_SIZE];
    strcpy(ride_name, "Thunderbolt");
    struct ride *roller_coaster = create_ride(ride_name, ROLLER_COASTER);

    // Print ride details
    printf("Ride created:\n");
    printf("  Name: %s\n", roller_coaster->name);
    printf("  Type: %d\n", roller_coaster->type);
    printf("  Rider Capacity: %d\n", roller_coaster->rider_capacity);
    printf("  Queue Capacity: %d\n", roller_coaster->queue_capacity);
    printf("  Minimum Height: %.2f\n", roller_coaster->min_height);
    if (roller_coaster->queue == NULL) {
        printf("  Queue: NULL\n");
    } else {
        printf("  Queue not set to NULL\n");
    }
    if (roller_coaster->next == NULL) {
        printf("  Next Ride: NULL\n\n");
    } else {
        printf("  Next Ride not set to NULL\n\n");
    }


    // Create a visitor
    char visitor_name[MAX_SIZE];
    strcpy(visitor_name, "Sasha");
    struct visitor *visitor = create_visitor(visitor_name, 160.5);

    // Print visitor details
    printf("Visitor created:\n");
    printf("  Name: %s\n", visitor->name);
    printf("  Height: %.2f cm\n", visitor->height);

    if (visitor->next == NULL) {
        printf("  Next Visitor: NULL\n\n");
    } else {
        printf("  Next Visitor not set to NULL\n\n");
    }

    return 0;
}
```

This code just calls `initialise_park` , `create_visitor` and `create_ride` to `malloc` and initialise, and then prints out all of their fields.

You can compile and run your program with the following command:

```
$ dcc cs_amusement_park.c main.c -o cs_amusement_park
$ ./cs_amusement_park
```

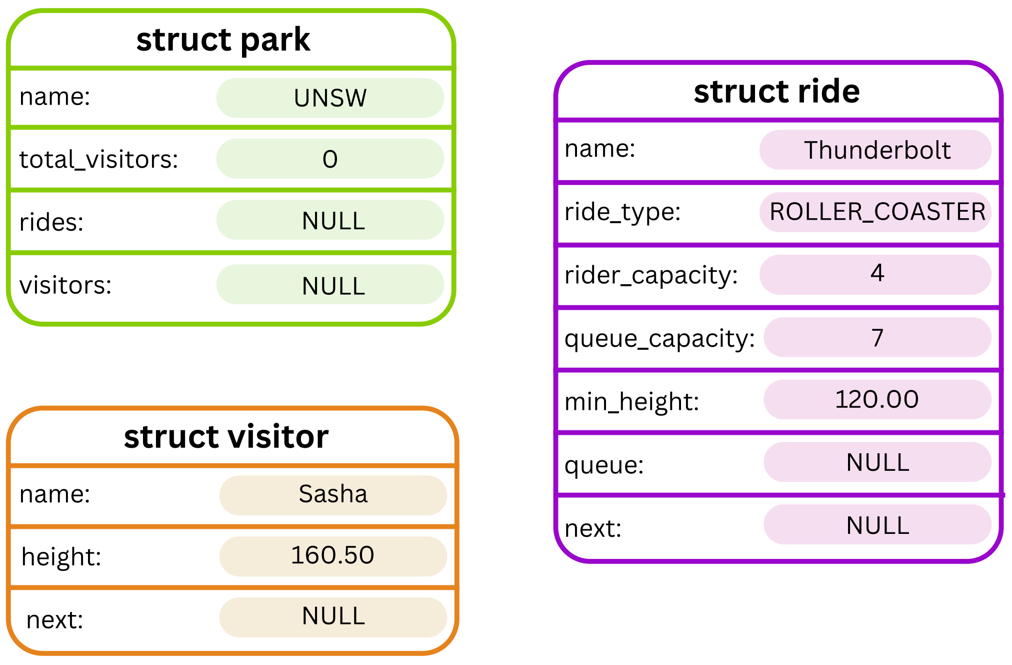
When you run it, it should print out something like:

```
Park initialised:
  Name: UNSW
  Total Visitors: 0
  Rides: NULL
  Visitors: NULL

Ride created:
  Name: Thunderbolt
  Type: 0
  Rider Capacity: 4
  Queue Capacity: 7
  Minimum Height: 120.00
  Queue: NULL
  Next Ride: NULL

Visitor created:
  Name: Sasha
  Height: 160.50 cm
  Next Visitor: NULL
```

Visually this can be represented as below:



WARNING:

Don't forget to delete any testing code you wrote before moving on to the next stage!

Stage 1.2 - Command loop and help

Now, we'll implement the command loop, allowing your program to take in and perform different operations on the park.

You'll find the following unimplemented function stub in `cs_amusement_park.c` in the starter code:

– Function Stub

```
// Stage 1.2
// Function to run the command loop
//
// Params:
//     park - the park
// Returns: None
void command_loop(struct park *park) {
    // TODO: Replace this with your code
    printf("Command loop not yet implemented\n");
    return;
}
```

This function is already called in the `main()` function in the `main.c` file for you.

From this stage onwards, your program should run in a loop, scanning in and executing commands until `CTRL+D` is entered. You should implement this command loop between the welcome and goodbye messages, so that your program runs in the following order:

1. First, print out the welcome message (the printf statement is included in the starter code).
2. Then run in a loop until `CTRL+D` , and:
    1. Print out the prompt `Enter command:`
    2. Scan in a command character
    3. Scan in any arguments following the command character
    4. Execute the command.
3. After `CTRL+D` is pressed, the goodbye message should be printed (the printf statement is included in the starter code), and then your program should end.

Each command will start with a single unique character and may be followed by a variable number of arguments depending on the command.
The unique character for each different command and the number and type of the command arguments are specified in the relevant stages of this assignment.

HINT:

This command loop is *extremely* similar to both:

- Stage 1.3 of Assignment 1,
- cs_calculator from the week 4 lab.

Just like in these activities, **Make sure to put a space before the `%c`** in `scanf(" %c", ...)` ,
so that it ignores all the preceding whitespace characters.

The first command you will implement is the **help** command, which is described below.

Command

```
?
```

Description

When the character `'?'` is entered into your program, it should run the **Help** command. This command doesn't read in any arguments following the initial command and should display a message which lists the different commands and their arguments.

The purpose of this command is to allow anyone using our program to easily look up the different commands.

A helper function: `print_usage()` in `cs_amusement_park.c` has been provided to help you print and format this message.

– Helper Function

```c
// Function to print the usage of the program
// '?' command
// Params: None
// Returns: None
void print_usage(void) {
    printf(
        "======================[ CS Amusement Park ]======================\n"
        "      ===============[     Usage Info     ]===============       \n"
        "  a r [ride_name] [ride_type]                                    \n"
        "    Add a ride to the park.                                      \n"
        "  a v [visitor_name] [visitor_height]                            \n"
        "    Add a visitor to the park.                                   \n"
        "  i [index] [ride_name] [ride_type]                              \n"
        "    Insert a ride at a specific position in the park's ride list.\n"
        "  j [ride_name] [visitor_name]                                   \n"
        "    Add a visitor to the queue of a specific ride.               \n"
        "  m [visitor_name] [ride_name]                                   \n"
        "    Move a visitor from roaming to a ride's queue.               \n"
        "  d [visitor_name]                                               \n"
        "    Remove a visitor from any ride queue and return to roaming.  \n"
        "  p                                                              \n"
        "    Print the current state of the park, including rides and     \n"
        "    visitors.                                                    \n"
        "  t                                                              \n"
        "    Display the total number of visitors in the park, including  \n"
        "    those roaming and in ride queues.                            \n"
        "  c [start_ride] [end_ride]                                      \n"
        "    Count and display the number of visitors in queues between   \n"
        "    the specified start and end rides, inclusive.                \n"
        "  l [visitor_name]                                               \n"
        "    Remove a visitor entirely from the park.                     \n"
        "  r                                                              \n"
        "    Operate all rides, allowing visitors to enjoy the rides      \n"
        "    and moving them to roaming after their ride.                 \n"
        "  S [ride_name]                                                  \n"
        "    Shut down a ride, redistributing its visitors to other rides.\n"
        "  M [ride_type]                                                  \n"
        "    Merge the two smallest rides of the specified type.          \n"
        "  s [n] [ride_name]                                              \n"
        "    Split an existing ride into `n` smaller rides.               \n"
        "  T [n] [command]                                                \n"
        "    Schedules a command to execute in `n` ticks                  \n"
        "  q                                                              \n"
        "    Quit the program and free all allocated resources.           \n"
        "  ?                                                              \n"
        "    Show this help information.                                  \n"
        "=================================================================\n");
}
```

This is what it should look like when you run the **Help** command:

– Example

```
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.
=================================================================
```

NOTE:

At this point in the assignment, the **ONLY** command you have to implement is the **Help** command, which prints this message.

None of the other commands listed here should be implemented yet.

Instead, you will be adding them over the course of the whole assignment.

Because of the command loop, after the **Help** command has finished running, your program should print out

```
Enter command:
```

and then wait for the user to either enter another command, or press `CTRL+D` to end the program.

Errors

- If an invalid command is entered, your program should print `ERROR: Invalid command.`

Clarifications

- All commands will begin with a single unique character.
  Some will be followed by a number of arguments, depending on the command.
- You can assume that commands will always be given the correct number of arguments, so you do **NOT** need to check for this.

Examples

– Example 1.2.1: Run no commands

Input:

```
UNSW
[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: [CTRL+D]
Goodbye!
```

+ Example 1.2.2: Help Command

NOTE:

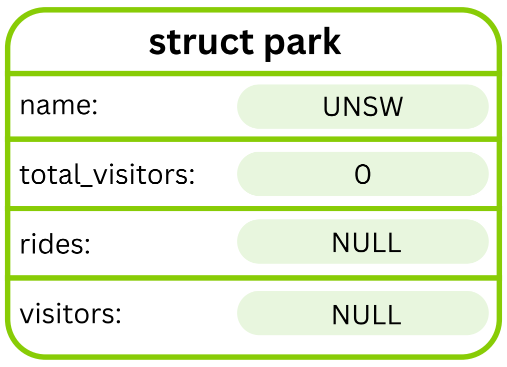
You may like to autotest this section with the following command:

```
1091 autotest-stage 01_02 cs_amusement_park
```

Stage 1.3 - Append rides and visitors

Now it's time to start building your amusement park!
When you first run your program, the park (and your linked list representing the rides and visitors) will start empty.

It should look something like this:



struct park	
name:	UNSW
total_visitors:	0
rides:	NULL
visitors:	NULL

To make your amusement park more interesting, we need a way of adding rides and visitors.

– Function Stubs

```c
// Stage 1.3
// Function to add a ride to the park
// Params:
//      p - the park
// Returns: None
void add_ride(struct park *park) {
    // TODO: Replace this with your code
    printf("Add ride not yet implemented\n");
    return;
}
```

```c
// Stage 1.3
// Function to add a visitor to the park
// Params:
//      p - the park
// Returns: None
void add_visitor(struct park *park) {
    // TODO: Replace this with your code
    printf("Add visitor not yet implemented\n");
    return;
}
```

Description

The `a` command appends a ride or visitor to the park, and will be followed by another character based on whether you are appending a ride `r` or a visitor `v` . Then scans in more information based on that more input is entered.

If you are appending a ride, `r` :

- a name and,
- a type

will also be scanned in.

If you are appending a visitor, `v` :

- a name and,
- a height

will also be scanned in.

Provided helper functions

Two helper functions have been provided for this stage:

- `enum ride_type scan_type(void)` :
  - Scans the `ride_type` and returns it as an `enum ride_type` .
  - Returns `INVALID` if the `ride_type` did not correspond to one of the valid ride types.
- `void scan_name(char name[MAX_SIZE])`
  - Scans the `name` into `char name[MAX_SIZE]` .

Command

```
a r [name] [type]
```

– Function Stub

```c
// Stage 1.3
// Function to add a ride to the park
// Params:
//      p - the park
// Returns: None
void add_ride(struct park *park) {
    // TODO: Replace this with your code
    printf("Add ride not yet implemented\n");
    return;
}
```

Description

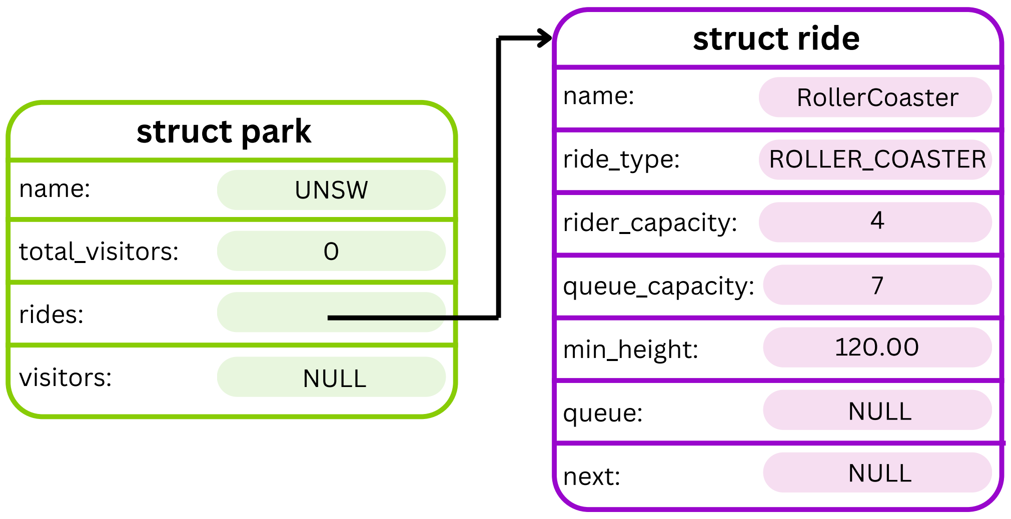
When this command is run, your program should create a new ride containing the name and type, assigning all other fields based on the type, and then **append** it to the end of the list of rides (i.e. perform a tail insertion).

Finally, it should print out a message to confirm that the command was successful:

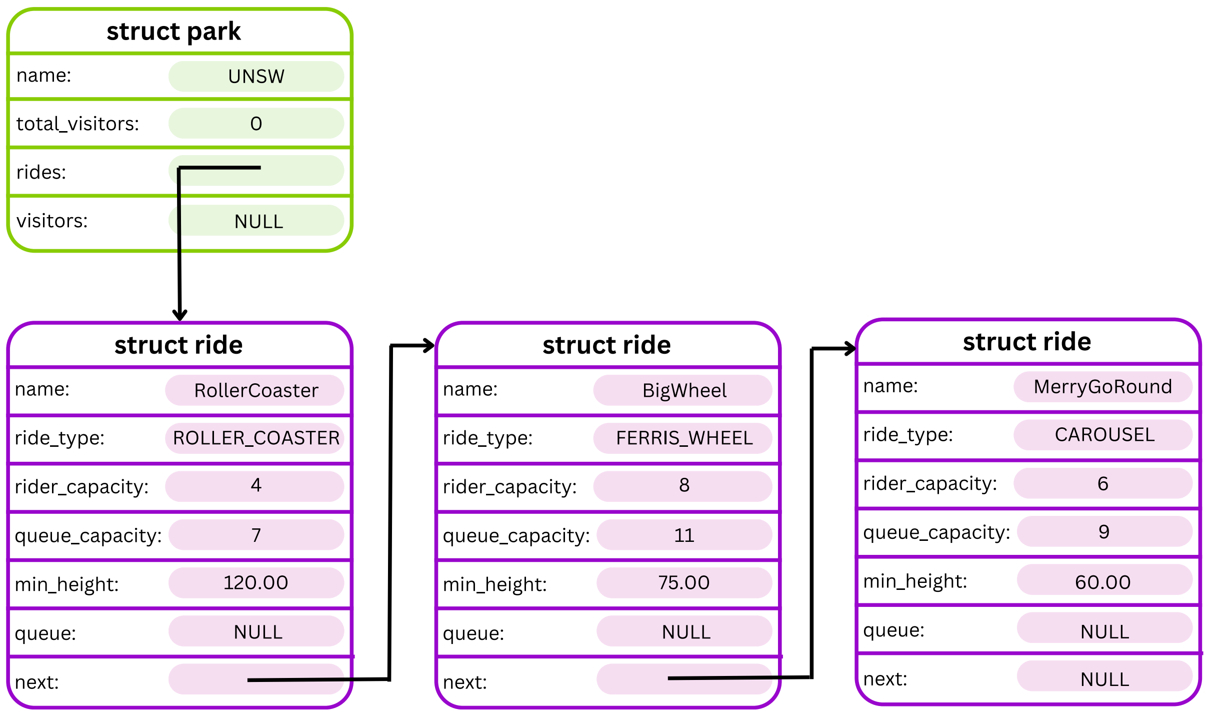
`Ride: '[name]' added!`

You should replace `[name]` in this message with the `name` you scanned in.

After appending one ride, your park should look something like this:



Any other rides added via this command will all be appended to the end of the list, i.e. a tail insertion, looking something like this:



Examples

– Example 1.3.1: Add one ride

Input:

```
UNSW
a r RollerCoaster ROLLER_COASTER
[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 RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' added!
Enter command: [CTRL+D]
Goodbye!
```

– Example 1.3.2: Add multiple rides

Input:

```
UNSW
a r MerryGoRound CAROUSEL
a r BigWheel FERRIS_WHEEL
a r SkyHigh ROLLER_COASTER
[CTRL+D]
```

Input and Output:

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

Enter the name of the park: UNSW
Enter command: a r MerryGoRound CAROUSEL
Ride: 'MerryGoRound' added!
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a r SkyHigh ROLLER_COASTER
Ride: 'SkyHigh' added!
Enter command: [CTRL+D]
Goodbye!
```

Command

```
a v [name] [height]
```

– Function Stub

```c
// Stage 1.3
// Function to add a visitor to the park
// Params:
//      p - the park
// Returns: None
void add_visitor(struct park *park) {
    // TODO: Replace this with your code
    printf("Add visitor not yet implemented\n");
    return;
}
```

Description

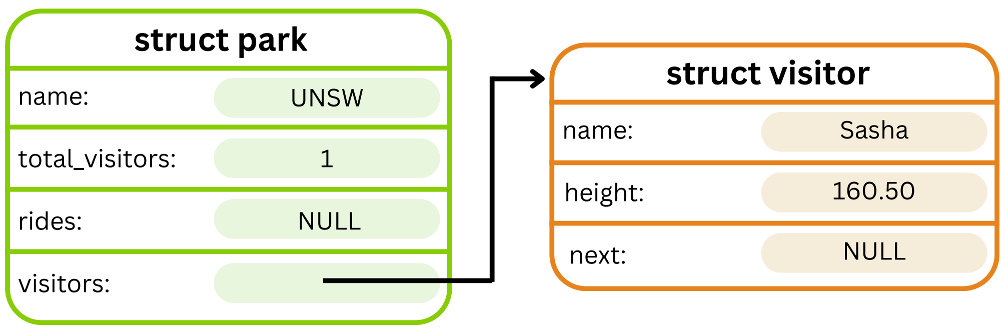
When this command is run, your program should create a new visitor containing the name and height, and then **append** it to the end of the list of visitors (i.e. perform a tail insertion).

Finally, it should print out a message to confirm that the command was successful:

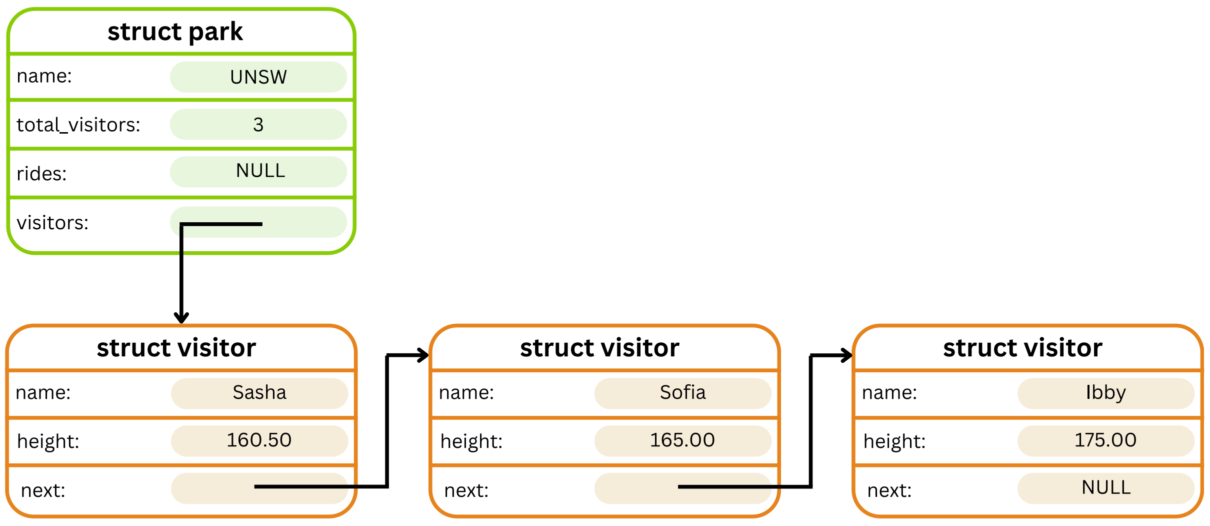
`Visitor: '[name]' has entered the amusement park!`

You should replace `[name]` in this message with the `name` you scanned in.

After appending one visitor, your park should look something like this:



Any other visitors added via this command will all be appended to the end of the list, i.e. a tail insertion, looking something like this:



Examples

Example 1.3.3: Add one visitor

Input:

```
UNSW
a v Sasha 160.5
[CTRL+D]
```

Input and Output:

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

Enter the name of the park: UNSW
Enter command: a v Sasha 160.5
Visitor: 'Sasha' has entered the amusement park!
Enter command: [CTRL+D]
Goodbye!
```

Example 1.3.4: Add multiple visitors

Input:

```
UNSW
a v Sasha 160.5
a v Sofia 165
a v Ibby 175
a v Grace 168.2
[CTRL+D]
```

Input and Output:

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

Enter the name of the park: UNSW
Enter command: a v Sasha 160.5
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 Grace 168.2
Visitor: 'Grace' has entered the amusement park!
Enter command: [CTRL+D]
Goodbye!
```

– Example 1.3.5: Add rides and visitors

Input:

```
UNSW
a r SwingSet CAROUSEL
a v Sasha 160.5
a r SkyHigh ROLLER_COASTER
a v Sofia 165
a r BigWheel FERRIS_WHEEL
a v Ibby 175
a v Grace 168.2
[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 SwingSet CAROUSEL
Ride: 'SwingSet' added!
Enter command: a v Sasha 160.5
Visitor: 'Sasha' has entered the amusement park!
Enter command: a r SkyHigh ROLLER_COASTER
Ride: 'SkyHigh' added!
Enter command: a v Sofia 165
Visitor: 'Sofia' has entered the amusement park!
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a v Ibby 175
Visitor: 'Ibby' has entered the amusement park!
Enter command: a v Grace 168.2
Visitor: 'Grace' has entered the amusement park!
Enter command: [CTRL+D]
Goodbye!
```

Clarifications

- No error handling is required in this stage.
- `type` will be entered as an uppercase string and automatically converted to the correct `enum ride_type` for you by the `scan_type` function when the function is used.
- `name` will always be less than `MAX_SIZE - 1`
- You don't need to worry about `INVALID` until **Stage 1.5**.
  Until then, you can assume that the returned `enum ride_type` will never be `INVALID` .

NOTE:

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

```
1091 autotest-stage 01_03 cs_amusement_park
```

Stage 1.4 - Printing the park

Command

```
p
```

– Function Stub

```c
// Stage 1.4
// Function to print the park
// Params:
//      p - the park
// Returns: None
void print_park(struct park *park) {
    // TODO: Replace this with your code
    printf("Print park not yet implemented\n");
    return;
}
```

Description

When the `p` command is run, your program should print out all rides in the amusement park, from head to tail and then all the visitors, from head to tail currently roaming the park.

A function `print_ride` has been provided for you to format and print a single ride:

– Provided helper function

```c
// Function to print a ride
// Params:
//      ride - the ride to print
// Returns: None
// Usage:
// ```
//      print_ride(ride);
// ```
void print_ride(struct ride *ride) {
    printf("  %s (%s)\n", ride->name, type_to_string(ride->type));
    printf("    Rider Capacity: %d\n", ride->rider_capacity);
    printf("    Queue Capacity: %d\n", ride->queue_capacity);
    printf("    Minimum Height: %.2lfcm\n", ride->min_height);
    printf("    Queue:\n");
    struct visitor *curr_visitor = ride->queue;
    if (curr_visitor == NULL) {
        printf("      No visitors\n");
    } else {
        while (curr_visitor != NULL) {
            printf(
                "      %s (%.2lfcm)\n", curr_visitor->name,
                curr_visitor->height);
            curr_visitor = curr_visitor->next;
        }
    }
}
```

Additionally, a helper function `print_welcome_message` has been provided to assist you.

– Provided helper function

```c
// Function to print a welcome message
// Params:
//      name - the name of the park
// Returns: None
// Usage:
// ```
//      print_welcome_message(name);
// ```
void print_welcome_message(char name[MAX_SIZE]) {
    printf("===================[ %s ]===================\n", name);
}
```

If there are no rides in the amusement park, the following message should be printed instead:

```
No rides!
```

If there are no visitors in the amusement park, the following message should be printed instead:

```
No visitors!
```

If there are no rides **and** no visitors in the amusement, the following message should be printed instead: `The amusement park is empty!`

Examples

– Example 1.4.1: Print one ride

Input:

```
UNSW
a r RollerCoaster ROLLER_COASTER
p
[CTRL+D]
```

Input and Output:

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

Enter the name of the park: UNSW
Enter command: a r RollerCoaster ROLLER_COASTER
Ride: 'RollerCoaster' added!
Enter command: p
===================[ UNSW ]===================
Rides:
  RollerCoaster (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
Visitors:
  No visitors!

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

– Example 1.4.3: Print one visitor

Input:

```
UNSW
a v Sasha 160.5
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 v Sasha 160.5
Visitor: 'Sasha' has entered the amusement park!
Enter command: p
===================[ UNSW ]===================
Rides:
  No rides!
Visitors:
  Sasha (160.50cm)

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

– Example 1.4.4: Print visitors and rides

Input:

```
UNSW
a r SwingSet CAROUSEL
a v Sasha 160.5
a r SkyHigh ROLLER_COASTER
a v Sofia 165
a r BigWheel FERRIS_WHEEL
a v Ibby 175
a v Grace 168.2
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 SwingSet CAROUSEL
Ride: 'SwingSet' added!
Enter command: a v Sasha 160.5
Visitor: 'Sasha' has entered the amusement park!
Enter command: a r SkyHigh ROLLER_COASTER
Ride: 'SkyHigh' added!
Enter command: a v Sofia 165
Visitor: 'Sofia' has entered the amusement park!
Enter command: a r BigWheel FERRIS_WHEEL
Ride: 'BigWheel' added!
Enter command: a v Ibby 175
Visitor: 'Ibby' has entered the amusement park!
Enter command: a v Grace 168.2
Visitor: 'Grace' has entered the amusement park!
Enter command: p
===================[ UNSW ]===================
Rides:
  SwingSet (CAROUSEL)
    Rider Capacity: 6
    Queue Capacity: 9
    Minimum Height: 60.00cm
    Queue:
      No visitors
  SkyHigh (ROLLER_COASTER)
    Rider Capacity: 4
    Queue Capacity: 7
    Minimum Height: 120.00cm
    Queue:
      No visitors
  BigWheel (FERRIS_WHEEL)
    Rider Capacity: 8
    Queue Capacity: 11
    Minimum Height: 75.00cm
    Queue:
      No visitors
Visitors:
  Sasha (160.50cm)
  Sofia (165.00cm)
  Ibby (175.00cm)
  Grace (168.20cm)

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

NOTE:

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

```
1091 autotest-stage 01_04 cs_amusement_park
```

Stage 1.5 - Handling Errors

In this stage, you will modify your stage 1.3 code to add some restrictions.

Error conditions

- When adding a ride, if the `type` (returned by the `scan_type` function) is `INVALID`, the following error should be printed: `ERROR: Invalid ride type.`
- When adding a ride, if there is already a ride in the park that contains the `name`, the following error should be printed. `ERROR: '[name]' already exists.`
- When adding a visitor, if there is already a visitor in the park that contains the `name`, the following error should be printed. `ERROR: '[name]' already exists.`
- When adding a visitor, if the height of visitor is `<` 50 or `>` 250, the following error should be entered `ERROR: Height must be between 50 and 250.`
- When adding a visitor, if the number of visitors in the park will exceed `40`, the following error should be entered `ERROR: Cannot add another visitor to the park. The park is at capacity.`.

NOTE:

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

Examples

– Example 1.5.1: Visitor Errors

Input:

```
UNSW
a v Grace 45
a v Grace 450
a v Grace 160
a v Grace 170
[CTRL+D]
```

Input and Output:

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

Enter the name of the park: UNSW
Enter command: a v Grace 45
ERROR: Height must be between 50 and 250.
Enter command: a v Grace 450
ERROR: Height must be between 50 and 250.
Enter command: a v Grace 160
Visitor: 'Grace' has entered the amusement park!
Enter command: a v Grace 170
ERROR: 'Grace' already exists.
Enter command: [CTRL+D]
Goodbye!
```

– Example 1.5.2: Ride Errors

Input:

```
UNSW
a r RollerCoaster INVALID
a r RollerCoaster CAROUSEL
a r RollerCoaster CAROUSEL
[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 RollerCoaster INVALID
ERROR: Invalid ride type.
Enter command: a r RollerCoaster CAROUSEL
Ride: 'RollerCoaster' added!
Enter command: a r RollerCoaster CAROUSEL
ERROR: 'RollerCoaster' already exists.
Enter command: [CTRL+D]
Goodbye!
```

– Example 1.5.3: Mixed Errors

Input:

```
UNSW
a r RollerCoaster INVALID
a r RollerCoaster CAROUSEL
a v Sofia 40
a v Sasha 150
a v Grace 270
a v Ibby 160
a r MerryGoRound CAROUSEL
[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 RollerCoaster INVALID
ERROR: Invalid ride type.
Enter command: a r RollerCoaster CAROUSEL
Ride: 'RollerCoaster' added!
Enter command: a v Sofia 40
ERROR: Height must be between 50 and 250.
Enter command: a v Sasha 150
Visitor: 'Sasha' has entered the amusement park!
Enter command: a v Grace 270
ERROR: Height must be between 50 and 250.
Enter command: a v Ibby 160
Visitor: 'Ibby' has entered the amusement park!
Enter command: a r MerryGoRound CAROUSEL
Ride: 'MerryGoRound' added!
Enter command: [CTRL+D]
Goodbye!
```

Clarifications

- If more than one error occurs, **only the first error in the order specified above** should be addressed by printing an error message. This is the same for all future commands.
- When checking if a ride or visitor name already exists in the park, the comparison is case-sensitive. For example, if a ride named `DPST1091` is already in the park, trying to add a ride named `Dpst1091` will not be treated as a duplicate, and no error will be raised. This is the same for all future commands.

NOTE:

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

```
1091 autotest-stage 01_05 cs_amusement_park
```

Testing and Submission

Remember to do your own testing

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

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

```
$ 1091 style cs_amusement_park.c
$ 1091 style cs_amusement_park.h
$ 1091 style main.c
$ 1091 autotest-stage 01 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!