

CS Dungeon

Overview



Your task in assignment 2 is to create a dungeon crawler game, where you will get to design a map of connected dungeons filled with monsters, items, and a final boss at the end to defeat! You will get to play as either a Fighter or a Wizard, using your special skills and stats to defeat these monsters and beat the game!

The game consists of a setup phase, where you add dungeons and items, and a gameplay phase, where you as the player will get to move between dungeons, fight monsters and collect items that can make you even stronger!

You can read about the history of dungeon crawler games [here](#).

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 a dungeon crawler game, where the dungeons are represented as a linked list, stored within a map. Each dungeon contains a list of items. The map also contains a player struct, which also contains a list of items.

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

NOTE:

There are 5 structs used in this assignment. You do not need to know everything from the beginning, each stage of the assignment will walk you through what you need to know. The easiest way to understand these structs is to get stuck into **Stage 1.1** where you get to create some!

– Structs

struct map

- Purpose: To store all the information about the dungeon map. It contains the list of dungeons within the map, the player, and the number of points required to win the game. The `entrance` field is a pointer to the first dungeon in the list of dungeons.
- Defined in `cs_dungeon.h`.

struct dungeon

- Purpose: To store all the information about a single dungeon. This will form a linked list of dungeons by pointing to the next one (or `NULL`).
- Defined in `cs_dungeon.c`.

struct item

- Purpose: To store all the information about a single item. This will form a linked list of items by pointing to the next one (or `NULL`). Items can be stored in a dungeon and in the player's inventory.

- Defined in `cs_dungeon.c` .

`struct player`

- Purpose: To store all the information about the player.
- Defined in `cs_dungeon.c` .

`struct boss`

- Purpose: To store all the information about the final boss.
- Defined in `cs_dungeon.c` .

The following enum definitions are also provided for you. You can create your own enums if you would like, but you should not modify the provided enums.

– Enums

`enum monster_type`

- Purpose: To represent the different types of monsters that can be encountered in the dungeons.
- Defined in `cs_dungeon.h` .

`enum item_type`

- Purpose: To represent the different types of items that can be found in the dungeons.
- Defined in `cs_dungeon.h` .

HINT:

Remember to initialise 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 Dungeon.

1. Create a new folder for your assignment work and move into it. You can follow the commands below to link and copy the files.

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

2. There are 3 files in this assignment. Run the following command below to download all 3, which will link the header file and the main file. This means that if we make any changes to `cs_dungeon.h` or `main.c` , you will not have to download the latest version as yours will already be linked.

```
$ 1091 fetch-activity cs_dungeon
```

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

```
$ 1091 autotest cs_dungeon
```

WARNING:

When running the autotest on the starter code (with no modifications), it is expected to see failed tests.

4. Read through the rest of the introductory specification and **Stage 1**.

Starter Code

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

- **Main File (`main.c`):** This file handles **all input scanning and error handling** for you. It also tests your code in `cs_dungeon.c` and contains the `main` function. You don't need to modify or fully understand this file, but if you're curious about how this assignment works, feel free to take a look. **You cannot change `main.c` .**
- **Header File (`cs_dungeon.h`):** contains defined constants that you can use and function prototypes. It also contains header comments that explain what functions should do and their inputs and outputs. *If you are confused about what a function should do, read the header file and the corresponding specification.* **You cannot change `cs_dungeon.h` .**
- **Implementation File (`cs_dungeon.c`):** contains stubs of functions for you to implement. This file does not contain a `main` function, so you will need to compile it alongside `main.c` . *This is the only file you may change. You do not need to use `scanf` or `fgets` anywhere.*

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_dungeon.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.**

WARNING:

Do not change the return type or parameter amount and type of the provided function stubs. `main.c` depends on these types and your code may not pass the autotests otherwise, as when testing we will compile your submitted `cs_dungeon.c` with the supplied `main.c` .

NOTE:

If you wish to create your own helper functions, you can put the function prototypes at the top of `cs_dungeon.c` and implement it later in the file. You should place your function comment just above the function definition.

How to Compile CS Dungeon

– Compiling CS Dungeon

To compile you should compile `cs_dungeon.c` alongside `main.c` . This will allow you to run the program yourself and test the functions you have written in `cs_dungeon.c` . Autotests have been written to compile your `cs_dungeon.c` with the provided `main.c` .

To compile your code, use the following command:

```
$ gcc cs_dungeon.c main.c -o cs_dungeon
```

Once your code is compiled, you can run it with the following command:

```
$ ./cs_dungeon
```

To autotest your code, use the following command:

```
$ 1091 autotest cs_dungeon
```

Reference Implementation

To help you understand the expected behaviour of CS Dungeon, we have provided a reference implementation. If you have any questions about the behaviour of your assignment, you can check and compare yours to the reference implementation.

To run the reference implementation, use the following command:

```
$ 1091 cs_dungeon
```

– Example Usage

Once you have followed the setup prompts, you might want to start by running the `?` command, whether you are in the setup phase or the gameplay phase.

When in the setup phase, the `?` command will display what you can add to the map:

```
$ 1091 cs_dungeon
Welcome to the 1091 Dungeon!
This is a game where you get to create your own dungeon map, battle monsters and collect items!
Please enter the name of your map: Faerun
Please enter the amount of points required to win: 25
Please enter the player's name: Minsc
Player Class Options:
    Fighter
    Wizard
Please enter player's chosen class type: Fighter

-----Setup Phase-----

Enter Command: ?
=====
=====[ 1091 Dungeon ]=====
=====[ Setup: Usage Info ]=====
?
    Show setup usage info
q
    Exit setup stage
a [dungeon name] [monster type] [num_monsters]
    Append a dungeon to the end of the map's list of dungeons
p
    Prints the map's list of dungeons
s
    Shows the player's current stats
i [position] [dungeon name] [monster type] [num_monsters]
    Inserts a dungeon to the specified position in the map
t [dungeon position] [item type] [points]
    Inserts an item into the specified dungeon's list of items
=====

Enter Command: Ctrl-D
Thanks for playing!
```

When in the gameplay phase, the `?` command will show you what actions the player can take:

```
$ 1091 cs_dungeon
Welcome to the 1091 Dungeon!
This is a game where you get to create your own dungeon map, battle monsters and collect items!
Please enter the name of your map: Faerun
Please enter the amount of points required to win: 25
Please enter the player's name: Minsc
Player Class Options:
    Fighter
    Wizard
Please enter player's chosen class type: Fighter

-----Setup Phase-----

Enter Command: a beach 1 1
beach has been added as a dungeon to the map!

Enter Command: q
Please enter the required item to defeat the final boss: 1
The final boss has been added to faerun!
Map of faerun!
|^|^|^|^|^|   |^|^|^|^|^|

1. beach
Boss: Present
Monster: Slime
Minsc is here

|^|^|^|^|^|   |^|^|^|^|^|

-----Gameplay Phase-----

Enter Command:?
=====
=====
?
  Show gameplay usage info
q
  Exit gameplay
p
  Prints the map's list of dungeons
s
  Shows the player's current stats
d
  Prints the current dungeon's details
>
  Move to the next dungeon in the map
<
  Move to the previous dungeon in the map
!
  Physically attack monsters in current dungeon
#
  Magically attack monsters in current dungeon
P
  Activate the player's class power
c [item number]
  Collect the specified item from current dungeon
u [item number]
  Use the specified item from the player's inventory
T
  Teleport to the furthest dungeon
b
  Fight the boss in the current dungeon
=====

Enter Command: Ctrl-D
Thanks for playing!
```

The easiest way to understand how this assignment works is to play a game yourself! Below is example input you can try by using the reference solution.

– Example Game

```
Faerun
25
Minsc
Fighter
a cavern 1 3
a castle 2 2
a graveyard 3 3
t 1 0 5
t 1 0 3
t 2 1 5
t 2 3 5
t 2 3 5
q
1
```

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/CPTG1391. 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 linked lists you will receive a 0 for performance in this assignment.**

Banned C Features

In this assignment, **you cannot use arrays for the list of dungeons nor the lists of items** and cannot use the features explicitly banned in the [Style Guide](#). If you use arrays in this assignment for the linked list of dungeons or the linked lists of items you will receive a **0** for performance in this assignment.

FAQ

– FAQ

Q: Can I edit the given starter code functions/structs/enums?

You can only edit `cs_dungeon.c` .You **cannot** edit anything in `main.c` or `cs_dungeon.h` .

You **cannot** edit any of the parameters or names for the provided functions.

Q: Can I use X other C feature

A: For everything not taught in the course, check the style guide. If it says "Avoid", then we may take style marks off if its not used correctly. If it says "Don't Use" then we will take style marks off (see the style marking rubric).

Game Structure

This game consists of a setup phase and a gameplay phase.

You will be implementing both the setup phase and gameplay phase throughout this assignment, adding more features to both as you progress. By the end of **stage 1.4** you will have implemented parts of both setup and gameplay enough to play a very basic game.

The game is ended either by the user entering `Ctrl-D` , the win condition being met, or the player running out of health points. The program can also be ended in the setup phase with `Ctrl-D` or `q` .

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.

NOTE:

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

- A non-existent command.
- 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` . **All scanning and most printing will be handled for you in `main.c` , so you should not need to worry about the above.**

[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 dungeon map and play a simple game!

Stage 1.1 Creating the Map, a Dungeon, and the Player

In Stage 1.1, you'll implement functions that allocate memory using `malloc` and initialise all fields for `struct map` , `struct dungeon` , and `struct player` .

You will find the following unimplemented function stubs in `cs_dungeon.c` :

– [Function Stubs](#)

```
struct map *create_map(char *name, int win_requirement) {
    // TODO: implement this function
    printf("Create Map not yet implemented.\n");
    exit(1);
}
```

```
struct dungeon *create_dungeon(char *name,
                                enum monster_type monster,
                                int num_monsters,
                                int contains_player) {
    // TODO: implement this function
    printf("Create Dungeon not yet implemented.\n");
    exit(1);
}
```

```
struct player *create_player(char *name, char *class_type) {
    // TODO: implement this function
    printf("Create Player not yet implemented.\n");
    exit(1);
}
```

NOTE:

`create_dungeon` is not called by `main.c` , so you are free to change the function if you wish.

In `main.c` , `create_map` is called after reading the map name and amount of points required to win, and `create_player` is called after reading the player name and class. `create_dungeon` will be called by you, in `cs_dungeon.c` , whenever a dungeon should be added to the map.

Your task is to implement the `create_map` function, so that it:

- 1. Creates a new `struct map` (using `malloc`).
- 2. Copies the `name` and `win_requirement` arguments into the corresponding struct fields.
- 3. Initialises all other fields to a reasonable value.
- 4. Returns a pointer to the newly created `struct map` .

You also then need to complete the `create_player` function, so that it:

- 1. Creates a new `struct player` (using `malloc`).
- 2. Copies the `name` and `class_type` arguments into the corresponding struct fields.
- 3. Initialises `health_points` , `shield_power` , `damage` and `magic_modifier` based on the chosen class (listed below).
- 4. Initialises all other fields to some reasonable value.
- 5. Returns a pointer to the newly created `struct player` .

– Class Stats

| Class | Fighter | Wizard |
|----------------|---------|--------|
| Health Points | 30 | 15 |
| Shield Power | 2 | 0 |
| Damage | 8 | 7 |
| Magic Modifier | 0.9 | 1.5 |

You also then need to complete the `create_dungeon` function, so that it:

- 1. Creates a new `struct dungeon` (using `malloc`).
- 2. Copies the `name` , `monster` , `num_monsters` and `contains_player` arguments into the corresponding struct fields.
- 3. Initialises all other fields to some reasonable value.
- 4. Returns a pointer to the newly created `struct dungeon` .

NOTE:

The `boss` field can be set to `NULL` when creating a dungeon. The final boss will be added in **Stage 1.5**.

Clarifications

- Initially there are no dungeons in the map.
- Initially there are no items in a dungeon, and no items in the player's inventory.
- No error handling is required for **Stage 1.1**.

Testing

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

Instead, you may want to double check your work by compiling your code using `dcc` and making sure there are no warnings or errors. Make sure that **every struct field** is initialised. If there are any errors in your code, they will become clear in **Stage 1.2** onwards.

– Optional Testing File

You can optionally use the file `cs_dungeon_stage1_testing.c` to test your functions. This file **should not** be submitted, it is only for basic testing, to ensure all your struct fields have been initialised.

This file **is not** used in testing when marking your submissions, do not submit it.

To fetch this file, use the following command:

```
$ 1091 fetch-testing cs_dungeon_stage1_testing
```

This testing file contains the line `#include "cs_dungeon."`, so that we can call your functions (`create_map` , `create_player` and `create_dungeon`) from `cs_dungeon.c` in this file. This means you do not need to edit this file (but can if you wish to)! Once you are happy with your code for these functions, you can test your implementation like below:

```
$ gcc cs_dungeon_stage1_testing.c cs_dungeon.c -o cs_dungeon_stage1_testing
$ ./cs_dungeon_stage1_testing
Map name: Faerun
Win requirement: 25

Player is NULL
Entrance is NULL

Player name: Minsc
Class type: Fighter
Damage: 8
Health points: 30
Shield power: 2
Magic modifier: 0.900000
Points: 0
Inventory is NULL

Dungeon name: forest
Number of monsters: 2
Contains player: 1
Dungeon contains skeletons.

Items is NULL
Boss is NULL
Next is NULL
```

Stage 1.2 Appending Dungeons

Now it's time to start building out your dungeon map! When you run your program, `create_map` and `create_player` will be called for you by `main.c`. Then, the setup phase loop will start, where you can begin to create your own custom dungeon map!

The first command you will be implementing is the Append Dungeon command.

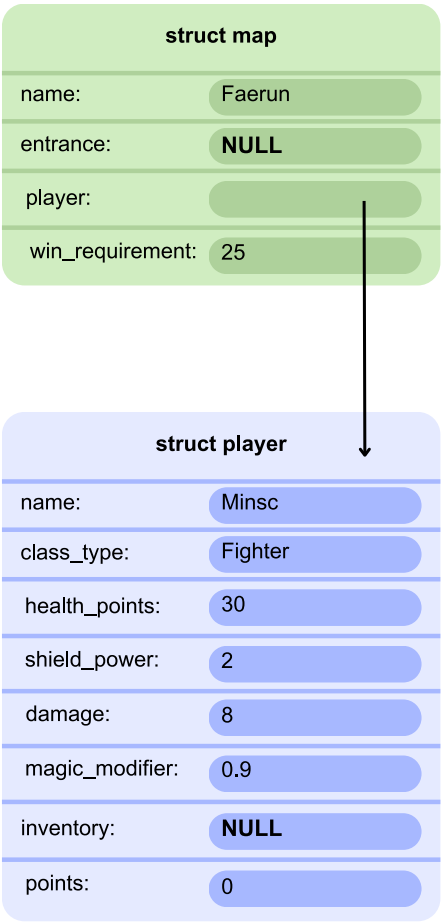
Command: Append Dungeon

Enter Command: `a [name] [monster] [num_monsters]`

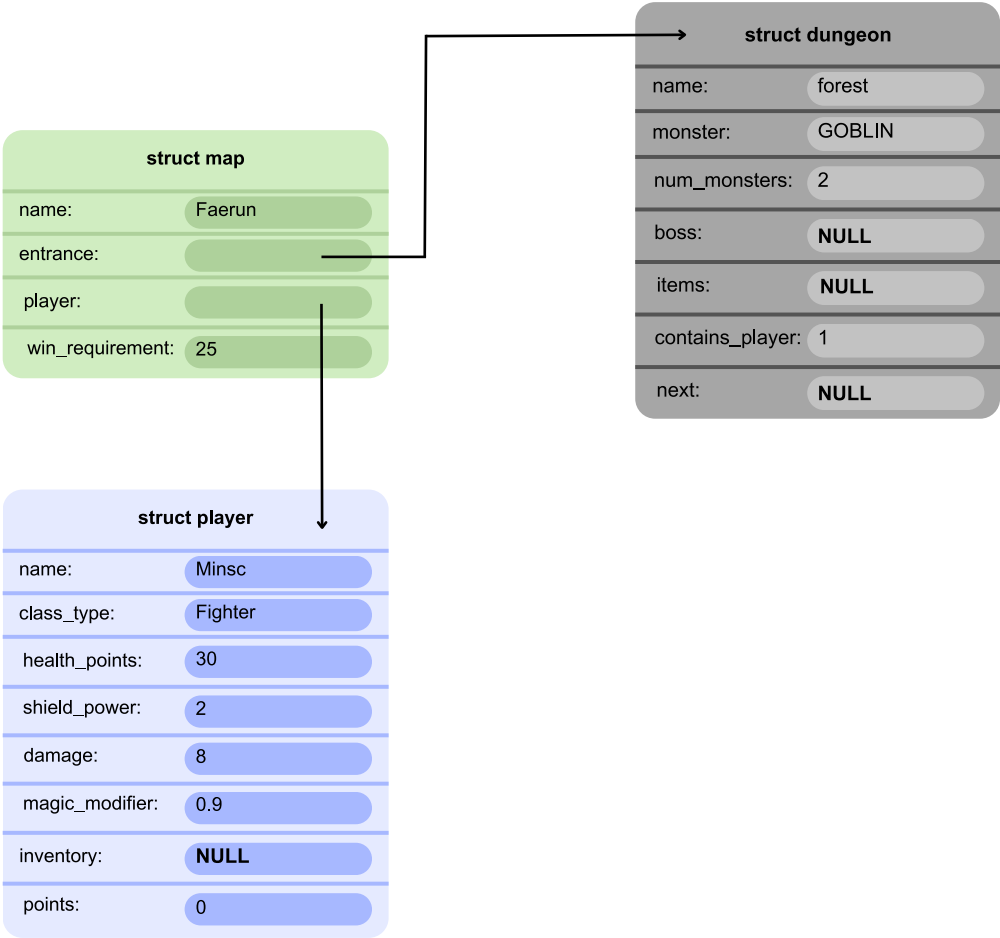
NOTE:

This input is all already scanned in for you in the command loop function (`command_loop`) located in `main.c`.

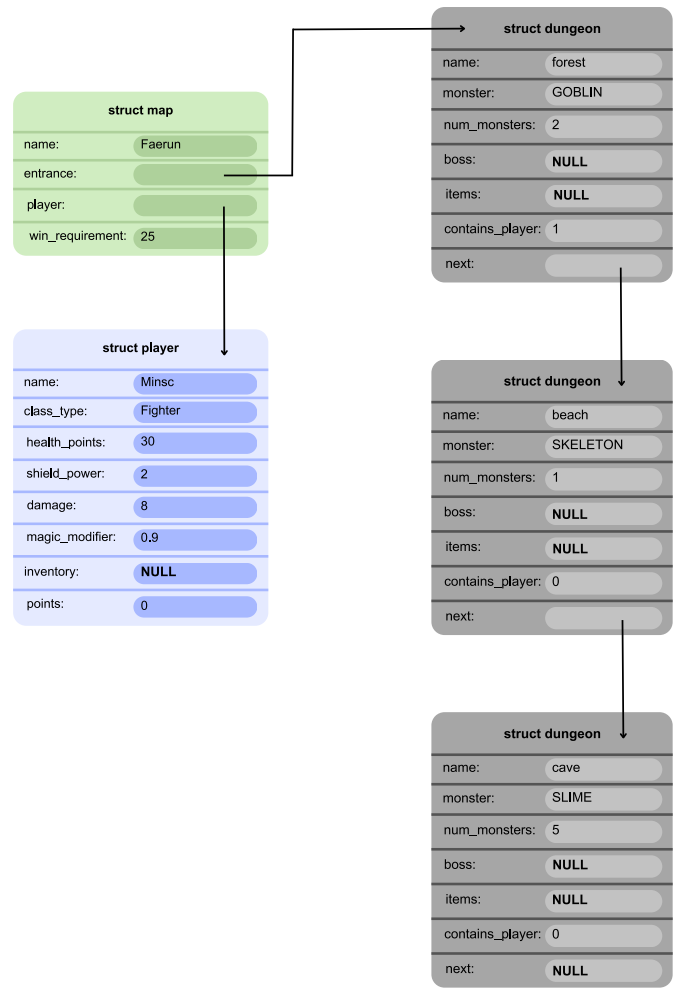
Before appending any dungeons, the map's list of dungeons will be empty, looking something like this:



After appending one dungeon, there will be one dungeon in the map, so that it looks something like this:



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



Your task is to implement the function `append_dungeon` in `cs_dungeon.c` which will append a newly created dungeon to the end of the map's list of dungeons, where `entrance` is the first dungeon in the list.

Additionally, the first dungeon in the map, i.e. what `entrance` points to, is where the player should start, so it should contain the player. If a dungeon contains the player, the `contains_player` field should be set to `1`. Otherwise, it should be set to `0`.

You will need to:

1. Create a new dungeon using your `create_dungeon` function from **Stage 1.1**, using the parameters provided.
2. Append the new dungeon to the end of the list of dungeons stored in the map, which starts at the `entrance` (head of the list), i.e. perform a tail insertion.

HINT:

Make sure to read the function prototype for `append_dungeon` in `cs_dungeon.h` for clear explanations regarding function parameters and their corresponding functionalities.

NOTE:

The `map` contains a pointer to the head of the list of dungeons, called `entrance`. This is the list you will append the new dungeon to.

This function returns an `int` - this value indicates whether or not the appending was successful or not. We will handle error checking in **Stage 1.5**, so for now you can return the constant `VALID` (defined in `cs_dungeon.h`). `main.c` will handle all input and output (including any `printf`, `scanf`, or `fgets` calls) for you already.

Clarifications

- The first dungeon in the map is where the player should begin.
- The `boss` field can be left as `NULL`. The final boss will be added in **Stage 1.5**.
- No error handling is required for **Stage 1.2**

Examples

– Example 1.2.1: Add one dungeon

```
$ gcc cs_dungeon.c main.c -o cs_dungeon
$ ./cs_dungeon
Welcome to the 1091 Dungeon!
This is a game where you get to create your own dungeon map, battle monsters and collect items!
Please enter the name of your map: Faerun
Please enter the amount of points required to win: 25
Please enter the player's name: Minsc
Player Class Options:
    Fighter
    Wizard
Please enter player's chosen class type: Fighter

-----Setup Phase-----

Enter Command: a sewer 1 3
sewer has been added as a dungeon to the map!

Enter Command: Ctrl-D
Thanks for playing!
```

– [Example 1.2.2: Add many dungeons](#)

```
$ gcc cs_dungeon.c main.c -o cs_dungeon
$ ./cs_dungeon
Welcome to the 1091 Dungeon!
This is a game where you get to create your own dungeon map, battle monsters and collect items!
Please enter the name of your map: UNSW
Please enter the amount of points required to win: 40
Please enter the player's name: Bob
Player Class Options:
    Fighter
    Wizard
Please enter player's chosen class type: Wizard

-----Setup Phase-----

Enter Command: a classroom 1 1
classroom has been added as a dungeon to the map!

Enter Command: a forest 2 2
forest has been added as a dungeon to the map!

Enter Command: a lab 3 3
lab has been added as a dungeon to the map!

Enter Command: Ctrl-D
Thanks for playing!
```

NOTE:

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

```
1091 autotest-stage 01_02 cs_dungeon
```

Stage 1.3 - Printing the Dungeon Map

Now we want to be able to display all dungeons in the map and their basic details.

Command: Print dungeons

Enter Command: **p**

When the `p` command is run, your program should print out all of the dungeons stored in the map and their basic information. This is each dungeon’s position (in order of head to tail, indexed from 1), name, if the boss is present, and if the player is currently in that dungeon. This command can be called in both the setup phase and the gameplay phase, `main.c` will handle this for you.

There is one function stub to implement for **Stage 1.3**, called `print_map` in `cs_dungeon.c` .

Four functions have been provided for you in `cs_dungeon.c` to help print the list of dungeons:

- `print_empty_map` : prints an empty map message.
- `print_map_name` : prints a message saying "Map of [name]".
- `print_basic_dungeon` : prints the basic details of a given dungeon.
- `print_connection` : prints a tunnel connecting adjacent dungeons.

NOTE:

The last dungeon should have no tunnel after it.

You will need to:

1. Print every dungeon in the list stored in the map, using the provided printing functions.

Clarifications

- The map should only be empty in the setup phase, before any dungeons have been added.
- No error handling is required for **Stage 1.3**

Examples

– Example 1.3.1: Print a map with one dungeon

```
$ gcc cs_dungeon.c main.c -o cs_dungeon
$ ./cs_dungeon
Welcome to the 1091 Dungeon!
This is a game where you get to create your own dungeon map, battle monsters and collect items!
Please enter the name of your map: UNSW
Please enter the amount of points required to win: 30
Please enter the player's name: Bob
Player Class Options:
    Fighter
    Wizard
Please enter player's chosen class type: Wizard

-----Setup Phase-----

Enter Command: a classroom 3 3
classroom has been added as a dungeon to the map!

Enter Command: p
Map of UNSW!
|^|^|^|^|^|   |^|^|^|^|^|

1. classroom
Boss: None
Monster: Skeleton
Bob is here

|^|^|^|^|^|   |^|^|^|^|^|

Enter Command: Ctrl-D
Thanks for playing!
```

- Example 1.3.2: Print a map with multiple dungeons

```
$ gcc cs_dungeon.c main.c -o cs_dungeon
$ ./cs_dungeon
Welcome to the 1091 Dungeon!
This is a game where you get to create your own dungeon map, battle monsters and collect items!
Please enter the name of your map: Faerun
Please enter the amount of points required to win: 25
Please enter the player's name: Minsc
Player Class Options:
    Fighter
    Wizard
Please enter player's chosen class type: Fighter

-----Setup Phase-----

Enter Command: a Sewers 1 3
Sewers has been added as a dungeon to the map!

Enter Command: a Desert 2 2
Desert has been added as a dungeon to the map!

Enter Command: a Forest 3 4
Forest has been added as a dungeon to the map!

Enter Command: a Town 4 1
Town has been added as a dungeon to the map!

Enter Command: p
Map of Faerun!
|^|^|^|^|^|   |^|^|^|^|^|

1. Sewers
Boss: None
Monster: Slime
Minsc is here

|^|^|^|^|^|   |^|^|^|^|^|
|           |
|           |
|           |
|^|^|^|^|^|   |^|^|^|^|^|

2. Desert
Boss: None
Monster: Goblin
Empty

|^|^|^|^|^|   |^|^|^|^|^|
|           |
|           |
|           |
|^|^|^|^|^|   |^|^|^|^|^|

3. Forest
Boss: None
Monster: Skeleton
Empty

|^|^|^|^|^|   |^|^|^|^|^|
|           |
|           |
|           |
|^|^|^|^|^|   |^|^|^|^|^|

4. Town
Boss: None
Monster: Wolf
Empty

|^|^|^|^|^|   |^|^|^|^|^|
```

Enter Command: `Ctrl-D`
Thanks for playing!

– Example 1.3.3: Print a map with no dungeons

```
$ gcc cs_dungeon.c main.c -o cs_dungeon
$ ./cs_dungeon
Welcome to the 1091 Dungeon!
This is a game where you get to create your own dungeon map, battle monsters and collect items!
Please enter the name of your map: UNSW
Please enter the amount of points required to win: 30
Please enter the player's name: Minsc
Player Class Options:
    Fighter
    Wizard
Please enter player's chosen class type: Fighter

-----Setup Phase-----

Enter Command: p
There are no dungeons currently in the dungeon.

Enter Command: Ctrl-D
Thanks for playing!
```

NOTE:

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

```
1091 autotest-stage 01_03 cs_dungeon
```

Stage 1.4 Adding the Final Boss

So that we can move on to the gameplay component of CS Dungeon, we need to add a boss-level monster to the final dungeon in the map.

When `q` command is used in the setup phase, the player will then be prompted to enter the item type required to defeat the final boss:

```
Please enter the required item to defeat the final boss: [item type]
```

Then, the function `final_boss` in `cs_dungeon.c` will be called, which you will need to implement in this stage. This function returns an `int` - this value indicates whether or not adding the final boss was successful or not. We will handle error checking in **Stage 1.5**, so for now you can return the constant `VALID` (defined in `cs_dungeon.h`).

The map should always have at least one dungeon in it when it is time to add the final boss, and throughout the gameplay phase (the one the player is in).

The final boss's stats are as follows:

| | |
|---------------|----|
| Health Points | 35 |
| Damage | 10 |
| Points | 20 |

There are two functions to implement in **Stage 1.4**, `create_boss` and `final_boss`. You will need to:

1. Create a `struct boss` (using `malloc`) similarly to the create functions in **Stage 1.1** in the `create_boss` function.
2. Add this final boss to the very last dungeon in the map, specifically to the dungeon's `boss` field in the `final_boss` function.

After you have completed this stage, you will be able to play a very basic game of CS Dungeon, where you can add dungeons, a final boss, and print the map!

Clarifications

- There will always be at least one dungeon in the before moving to the gameplay phase (`main.c` checks for this).
- No error handling is required for **Stage 1.4**.

Examples

- [Example 1.4.1: Map with many dungeons, add a final boss](#)

```
$ dcc cs_dungeon.c main.c -o cs_dungeon
$ ./cs_dungeon
Welcome to the 1091 Dungeon!
This is a game where you get to create your own dungeon map, battle monsters and collect items!
Please enter the name of your map: Faerun
Please enter the amount of points required to win: 25
Please enter the player's name: Minsc
Player Class Options:
    Fighter
    Wizard
Please enter player's chosen class type: Fighter

-----Setup Phase-----

Enter Command: a Sewers 1 3
Sewers has been added as a dungeon to the map!

Enter Command: a Desert 2 2
Desert has been added as a dungeon to the map!

Enter Command: a Forest 3 4
Forest has been added as a dungeon to the map!

Enter Command: a Town 4 1
Town has been added as a dungeon to the map!

Enter Command: q
Please enter the required item to defeat the final boss: 0
The final boss has been added to Faerun!
Map of Faerun!
|^|^|^|^|^|   |^|^|^|^|^|

1. Sewers
Boss: None
Monster: Slime
Minsc is here

|^|^|^|^|^|   |^|^|^|^|^|
      |      |
      |      |
      |      |
|^|^|^|^|^|   |^|^|^|^|^|

2. Desert
Boss: None
Monster: Goblin
Empty

|^|^|^|^|^|   |^|^|^|^|^|
      |      |
      |      |
      |      |
|^|^|^|^|^|   |^|^|^|^|^|

3. Forest
Boss: None
Monster: Skeleton
Empty

|^|^|^|^|^|   |^|^|^|^|^|
      |      |
      |      |
      |      |
|^|^|^|^|^|   |^|^|^|^|^|

4. Town
Boss: Present
Monster: Wolf
Empty
```

|^|^|^|^|^| |^|^|^|^|^|

-----Gameplay Phase-----

Enter Command: Ctrl-D

Thanks for playing!

NOTE:

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

1091 autotest-stage 01_04 cs_dungeon

Stage 1.5 Handling Errors

Now we need to make sure that when setting up our map, that any invalid dungeon or boss inputs are handled correctly. From **Stage 1.5** onwards, you will need to do error checking to ensure correct inputs when creating a dungeon and will need to edit your code from earlier stages.

You will need to check for the following errors in your `append_dungeon` and `final_boss` functions:

- Invalid name,
- Invalid monster type,
- Invalid monster amount,
- Invalid required item,

and if they are, return the corresponding constant. You’re currently returning `VALID` by default.

The error constants are defined in `cs_dungeon.h`, and you can refer to the header comments for `append_dungeon` and `final_boss` for which constants to return in each case.

To check whether each field is valid or not, you must perform the following checks **in the below order**:

Appending dungeons

- `name` is invalid if it is already being used by another dungeon in the map.
- `monster` is invalid if it is not one of `SLIME`, `GOBLIN`, `SKELETON` or `WOLF`.
- `num_monsters` is invalid if it is not between 1 and 10 (inclusive).

NOTE:

If a dungeon is invalid, it should not be added to the list of dungeons in the map.

Creating the final boss

- `required_item` is invalid if it is not one of `PHYSICAL_WEAPON`, `MAGICAL_TOME`, `ARMOR`, `HEALTH_POTION` or `TREASURE`.

NOTE:

If the required item when making the final boss is invalid, `main.c` will keep asking for a new required item until it is valid.

HINT:

The function `strcmp` will be useful to check if `name` has been used before.

You can see how to use `strcmp` from the built-in Linux manual pages by typing `man 3 strcmp` into your terminal.

Examples

- Example 1.5.1: Attempting to append invalid dungeons and boss required item

```
$ gcc cs_dungeon.c main.c -o cs_dungeon
$ ./cs_dungeon
Welcome to the 1091 Dungeon!
This is a game where you get to create your own dungeon map, battle monsters and collect items!
Please enter the name of your map: Faerun
Please enter the amount of points required to win: 25
Please enter the player's name: Minsc
Player Class Options:
    Fighter
    Wizard
Please enter player's chosen class type: Fighter

-----Setup Phase-----

Enter Command: a sewer 10 5
ERROR: Invalid monster type. Monster type should be one of:
    1: SLIME
    2: GOBLIN
    3: SKELETON
    4: WOLF

Enter Command: a sewer 1 100
ERROR: Invalid monster number. Monster number must be between 1 and 10.

Enter Command: a sewer 1 3
sewer has been added as a dungeon to the map!

Enter Command: a sewer 3 1
ERROR: Invalid name. This name already exists in the map.

Enter Command: a beach 100 100
ERROR: Invalid monster type. Monster type should be one of:
    1: SLIME
    2: GOBLIN
    3: SKELETON
    4: WOLF

Enter Command: q
Please enter the required item to defeat the final boss: -10
ERROR: Invalid item type. Item type should be one of:
    0: PHYSICAL WEAPON
    1: MAGICAL TOME
    2: ARMOR
    3: HEALTH POTION
    4: TREASURE

Please enter the required item to defeat the final boss: 1
The final boss has been added to Faerun!
Map of Faerun!
|^|^|^|^|^|   |^|^|^|^|^|

1. sewer
Boss: Present
Monster: Slime
Minsc is here

|^|^|^|^|^|   |^|^|^|^|^|

-----Gameplay Phase-----

Enter Command: Ctrl-D
Thanks for playing!
```

NOTE:

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

```
1091 autotest-stage 01_05 cs_dungeon
```

Stage 1.6 - Checking Player Stats

We need a way to check what the player’s statistics are while we are playing our game. So in **Stage 1.6** you will be implementing a feature that checks the player's current stats, specifically:

- what dungeon they are in,
- their health points,
- shield power,
- damage,
- magic modifier,
- points collected,
- and what items they have in their inventory.

Command: Check Player Stats

```
Enter Command: s
```

There is one function to implement for **Stage 1.6**, called `player_stats` in `cs_dungeon.c` .

There are 3 helper functions provided for you in `cs_dungeon.c` to help you print everything:

- `print_player` : prints the details about the player.
- `print_no_items` : if there are no items in the player's inventory, this should be printed.
- `print_item` this prints the details of a given item. This does not need to be used until **Stage 3**, when items are added.

You will need to:

1. Find the above listed information about the player.
2. Use the provided printing functions to print the player's stats.

Clarifications

- If the map is empty e.g. in the setup phase before adding any dungeons, then `NULL` should be given as the dungeon name to `print_player` .
- Items are not added until **Stage 3**, so you do not need to use `print_item` until then. Instead, use `print_no_items` for now.
- There is no error handling in **Stage 1.6**.

Examples

- [Example 1.6.1: Checking player stats in gameplay phase](#)

```
$ dcc cs_dungeon.c main.c -o cs_dungeon
$ ./cs_dungeon
Welcome to the 1091 Dungeon!
This is a game where you get to create your own dungeon map, battle monsters and collect items!
Please enter the name of your map: Faerun
Please enter the amount of points required to win: 25
Please enter the player's name: Minsc
Player Class Options:
    Fighter
    Wizard
Please enter player's chosen class type: Fighter

-----Setup Phase-----

Enter Command: a Sewers 1 3
Sewers has been added as a dungeon to the map!

Enter Command: q
Please enter the required item to defeat the final boss: 0
The final boss has been added to Faerun!
Map of Faerun!
|^|^|^|^|^|   |^|^|^|^|^|

1. Sewers
Boss: Present
Monster: Slime
Minsc is here

|^|^|^|^|^|   |^|^|^|^|^|

-----Gameplay Phase-----

Enter Command: s
=====Player Stats=====
Minsc is currently in Sewers
Fighter
Health Points: 30
Shield Power: 2
Damage: 8
Magic Modifier: 0.9
Points Collected: 0
Minsc has the following items in their inventory:
No Items

Enter Command: Ctrl-D
Thanks for playing!
```

– Example 1.6.2: Checking player stats in setup phase

```
$ dcc cs_dungeon.c main.c -o cs_dungeon
$ ./cs_dungeon
Welcome to the 1091 Dungeon!
This is a game where you get to create your own dungeon map, battle monsters and collect items!
Please enter the name of your map: UNSW
Please enter the amount of points required to win: 30
Please enter the player's name: Minsc
Player Class Options:
    Fighter
    Wizard
Please enter player's chosen class type: Fighter

-----Setup Phase-----

Enter Command: s
=====Player Stats=====
Minsc has not entered the map yet!
Fighter
Health Points: 30
Shield Power: 2
Damage: 8
Magic Modifier: 0.9
Points Collected: 0
Minsc has the following items in their inventory:
No Items
Enter Command: Ctrl-D
Thanks for playing!
```

NOTE:

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

```
1091 autotest-stage 01_06 cs_dungeon
```

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_dungeon.c
$ 1091 autotest-stage 01 cs_dungeon
$ give dp1091 ass2_cs_dungeon cs_dungeon.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.

If you have not shared your assignment, you will not be penalised if your work is taken without your consent or knowledge.

For more information, read the [UNSW Student Code](#), or contact [the course account](#). 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. |

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_dungeon` .

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_dungeon cs_dungeon.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_dungeon.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 has been meaningfully decomposed into functions a maximum 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** (2025-04-11 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

(2025-03-19 09:00)

- Assignment Released

DPST1091/CPTG1391 25T1: Programming Fundamentals!