Spike: Task 10
Title: Game Data Structures
Author: Sam Huffer, 101633177

Goals / deliverables:

- A short research report evaluating data structures that could be used to create the player's inventory.
- A working inventory system using the chosen data structure, demonstrating access, addition and removal.

Technologies, Tools, and Resources used:

- Visual Studio 2017
- Microsoft Word
- Draw.io

Tasks undertaken

- Considered what features the data structure underlying an inventory would need to have.
- Looked at lecture notes and conducted research on data structures that could be used, noting down their features in relation to the criteria.
- Selected a data structure to form the basis of player inventories and general storage containers.
- I created the Item and Container classes, Item holding strings of information about an item, and Container building on top of vectors to allow players to store items, retrieve them and view them. I also created the ContainerItem class which inherits from both Item and Container to allow for items that are also containers, such as bags. Once the classes were complete, I made Location inherit from Container and added into the test World's starting location some test items.
- I added the Player class (which inherits from Container to allow for a player inventory), and began on look command functionality so that the player could look at items at their location or in their inventory. I started adding functionality for players to be able to look at items inside container items at their location or in their inventory, but ran into issues accessing the contents of the container item the player was looking into. I noticed that when I was getting that item from the location or player, I was casting it from the type Item to the type Container rather than Container Item. Once I changed that, the game then allowed me to look at the contents of container items.
- I considered how to implement the commands "take from", "put in" and "drop"; I originally implemented Container.ViewItem() to be able to get an "item in container in container in . . .", but I figured that would be fiddley to implement with the take command, so I cut back on that in the look command to only be able to access items in a container in the player's inventory or their current location.
- I implemented the "take from" command, allowing players to take items out of their current location or a container in their inventory or current location, and deposit said items into their inventory. Initially I only let players specify "take __ from __", but then added "take __ from __ in [inventory or location]". However, I later figured that was beyond the scope of this task, and removed it to keep it consistent with the look command.
- I added the "drop" command, allowing players to drop items in their inventory into their current location. I also included the ability to drop items currently in a container item in their inventory, into their current location, but commented it out for the moment.
- I added the "put __ in __" command, allowing players to take an item from their location or inventory and put it into a specified container at their location or in their inventory.
- I updated the Help stage to list the newly implemented commands that interact with the inventory and the player's location. While doing so, I commented in some alternative wordings of commands

that could be implemented in a future iteration, as well as a couple of possible expansions upon commands.

What we found out

- Some access methods (search, random reads) can be costly when applied to linked lists, as the list has to be traversed to get a result.
- The standard library includes singly-linked and doubly-linked lists (forward_list and list respectively), which I originally assumed programmers would have to create manually if they wanted to use them.
- The standard library includes multimaps: a map variant that allows key-value pairs to have identical keys.
- If class A inherits class B and C and is stored as class B, casting it as class C won't work; you have to cast it as class A to access class C members.
- The logic behind "put __ in __", "take __ [from __]" and "drop __" is similar, but is more different than just swapping out words in if statements.