Spike: 10
Title: Game Data Structures

Author: Liam Whitehouse, 103862481

Goals / deliverables:
The goal of this report is to research and explain four different data structures that could be used for a games inventory and exploring their pros and cons.

Technologies, Tools, and Resources used:
The tools I used to research this was a mixture of:
- Google.
- My own knowledge.

Tasks undertaken:
There are multiple different types of data structures that exist, just some are:
- Linked Lists.
- Vector Arrays.
- Stacks.
- Sets.
- Arrays.
- HashMap.

In our examples, we will be exploring the arrays that are more likely to be used in video games. These are:
- Vector Arrays:
    - This array is versatile and extremely useful for creating large inventories due to its ability to double itself in size when its about to hit its maximum capacity. However, its downfall is due to its size, it is very inefficient when searching for an element of when it needs to perform an insertion as it needs to loop through the entire array until it hits to where it needs, then move everything back one which can be taxing depending on the size of the array.
- HashMap:
    - HashMap are very efficient at insertions, lookups and deletions, this is due to the HashMap using a key-value pairs. However, the cons of this data structure are the ordering and possibility of keys having the same index.
- Linked-Lists:
    - Linked Lists is a data structure that makes use of pointers which are used to point to the next element in the array. These arrays can easily grow and shrink as their do not require re-sizing or reallocating memory. It is very easy to delete and insert elements in this array as well. This array however is not the best at

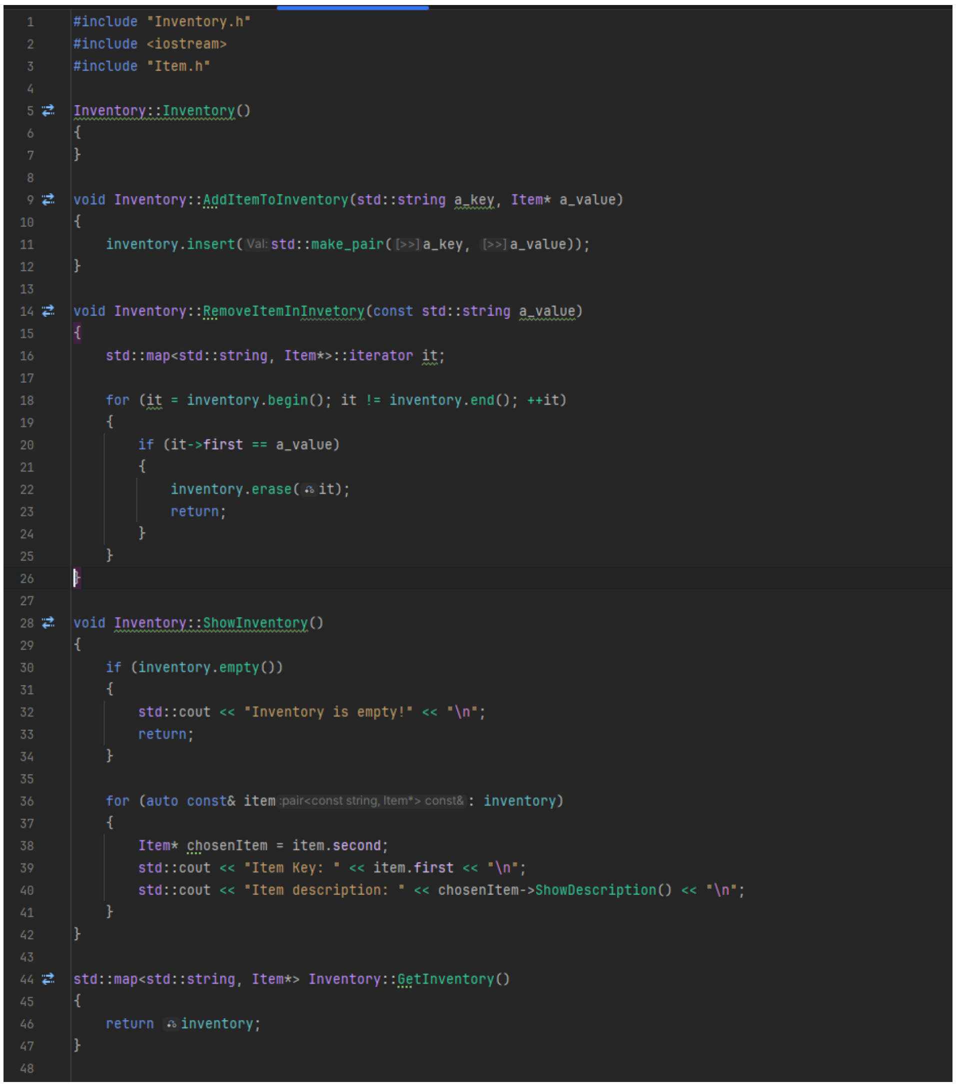
searching for elements in the array and it does add a layer of complexity due to it being built upon the usage of pointers.

- Arrays:
    - This array is the simplest out of the other arrays, the pros to this array is it is fast to access elements, simple to code with and has efficient iteration. The cons to this array is that it is a fixed size, so to increase its size we need to make a whole new array of that updated size and push all the elements into it and it can be costly to perform constant insertions and deletions.

  ss

What we found out:

For the games inventory I have decide to use a Hashmap. This is due to the Key and Value look-up, due to the game being a text-based adventure, it made sense to easily search for their values rather than looping through an entire array from start to finish.



```cpp
#include "Inventory.h"
#include <iostream>
#include "Item.h"

Inventory::Inventory()
{
}

void Inventory::AddItemToInventory(std::string a_key, Item* a_value)
{
    inventory.insert(std::make_pair(a_key, a_value));
}

void Inventory::RemoveItemInInvetory(const std::string a_value)
{
    std::map<std::string, Item*>::iterator it;

    for (it = inventory.begin(); it != inventory.end(); ++it)
    {
        if (it->first == a_value)
        {
            inventory.erase(it);
            return;
        }
    }
}

void Inventory::ShowInventory()
{
    if (inventory.empty())
    {
        std::cout << "Inventory is empty!" << "\n";
        return;
    }

    for (auto const& item : inventory)
    {
        Item* chosenItem = item.second;
        std::cout << "Item Key: " << item.first << "\n";
        std::cout << "Item description: " << chosenItem->ShowDescription() << "\n";
    }
}

std::map<std::string, Item*> Inventory::GetInventory()
{
    return inventory;
}
```