Spike: Task 9.P
Title: Game Data Structures

Author: Mitchell Wright, 100595153

Goals / deliverables:

To demonstrate an understanding of collection types, their strengths and weaknesses, and apply that knowledge in creating a functional inventory system.

Items created during task:

- Code, see: \09 - Spike – Game Data Structures\inventory\
- Report, see: \09 - Spike – Game Data Structures

Technologies, Tools, and Resources used:

- Visual Studio 2022
- SourceTree
- GitHub
- Lecture 2.2 – Game States & Stages

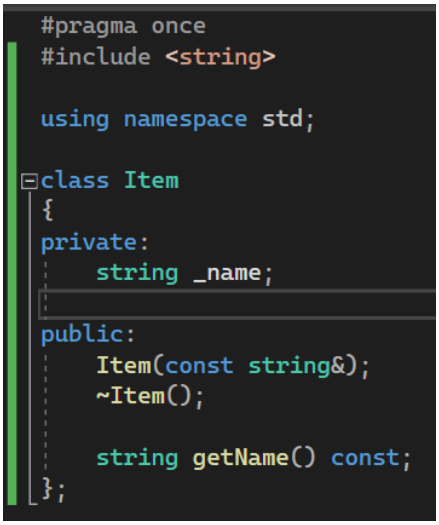
Tasks undertaken:

- Implementing Items
- Implementing the Inventory Class
- Implementing Viewing, Finding and Getting
- Implementing Addition and Removal
- Testing
- Commit to Git

What we found out:

1. Implementing Items:

   An inventory needs items. I decided, then, that it would be a good idea to make an item. Here's how.



```
#pragma once
#include <string>

using namespace std;

class Item
{
private:
    string _name;

public:
    Item(const string&);
    ~Item();

    string getName() const;
};
```

   Items basically just have a name, constructor and getter. There's nothing particularly special about this class, just a placeholder for future task.

2. Implementing the Inventory Class:

   As per the brief, the inventory needed to support adding items, removing items, and viewing items. I did, however, decide to add onto this with finding (necessary for addition and removal) and getting (in case the item ever needs to be retrieved from the inventory) for later tasks involving Zorkish so that I have a head start. I'm going to be a bit upset if it was all for naught.

```
class Inventory
{
private:
    vector<Item*> _items;

public:
    Inventory();
    ~Inventory();

    bool find(const string&);

    void view();

    bool add(Item*);
    bool remove(const string&);

    Item* get(const string&);
};
```

Using both the forbidden answer generator and my work from previous units, I tidied up things and added const where needed to avoid unnecessary copying. This was something I'd needed to be reminded of since I last worked with C++. Something noteworthy is my choice to use a vector instead of a map. This is partially due to my implementation of the items, with the names contained in them. Additionally, aside from vectors dynamically sized and efficient, the requirements of the inventory aren't particularly complex. Adding, removing, etc., don't require all that much coding and the basic checks needed to prevent errors are pretty basic too.

3. Implementing Viewing, Finding and Getting:

The simplest of the implementations in this seciton was viewing. Though I am tempted to go back and do it differently, making use of Ostream objects rather than letting the inventory handle it's own IO. This is an approach I might take later on in the unit.

```
void Inventory::view()
{
    for (Item* item : _items)
        cout << item->getName() << endl;
}
```

As shown above, it's a simple for each loop, and a cout print. Nothing special.
In the case of checking if an item exists in the inventory, the below code uses another for each loop, checking if the item name matches the input name.

```
bool Inventory::find(const string& name)
{
    for (Item* item : _items)
    {
        if (name == item->getName())
        {
            return true;
        }
    }
    return false;
}
```

The reason I've done it like this, is due to using a vector for the items. Had I used a map, I could have also used the built in find function, however I like this.

```
Item* Inventory::get(const string& name)
{
    for (Item* item : _items)
    {
        if (name == item->getName())
            return item;
    }

    return new Item("Error");
}
```

Getting works much the same, though rather than returning a Boolean, it returns the item or an item with name "Error".

4. Implementing Addition and Removal:

Addition makes use of the push_back function provided by the vector, with a couple of protections and a boolean return.

```
bool Inventory::add(Item* item)
{
    if (item != nullptr && !find(item->getName()))
    {
        _items.push_back(item);
        return true;
    }
    return false;
}
```

If the parameter fed through to the function is a nullptr or already exists within the inventory, then it isn't added. This avoids duplicates with the same name.

Removal is a bit more complicated and once again, was optimised by the forbidden answer generator.

```
bool Inventory::remove(const string& name)
{
    for (auto iter = _items.begin(); iter != _items.end(); ++iter)
    {
        if ((*iter)->getName() == name)
        {
            _items.erase(iter);
            return true;
        }
    }
    return false;
}
```

This method uses an iterator to loop through the items, then once the desired item is found, the item is removed and a bool is returned. Please also note the use of auto here and not in any previous part of the program. This is due to previous loops using pointers which supposedly may lead to unnecessary copies being made (according to VS2022's warning messages).

5. Testing:

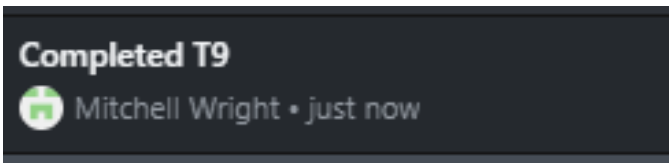
The test harness basically just goes through each of the above functions, testing a success state and fail state, as well as an edge cases at the end.

```
Inventory* inv = new Inventory();

inv->add(new Item("fork"));
inv->add(new Item("knife"));
inv->add(new Item("plate"));
inv->add(new Item("smaller plate"));

cout << "Testing Adding" << endl;
cout << inv->add(new Item("gun")) << endl;
cout << inv->add(new Item("gun")) << endl;

cout << "Testing Removing" << endl;
cout << inv->remove("gun") << endl;
cout << inv->remove("gun") << endl;


cout << "Testing Finding" << endl;
cout << inv->find("plate") << endl;
cout << inv->find("gun") << endl;

cout << "Testing Getting" << endl;
cout << inv->get("plate")->getName() << endl;
cout << inv->get("gun")->getName() << endl;

cout << "Testing Viewing" << endl;
inv->view();

cout << "Testing Edge Cases" << endl;
cout << inv->add(nullptr) << endl;
```

The program then prints these results.

```
Testing Adding
1
0
Testing Removing
1
0
Testing Finding
1
0
Testing Getting
plate
Error
Testing Viewing
fork
knife
plate
smaller plate
Testing Edge Cases
0
```

6. Commit History:



Sorta just completed this one in a single sitting.