Spike: 4
Title: Inventory

Author: Ryan Chessum, 102564760

Goals / deliverables:
- Code, see /09 – Spike - Game Data Structures/Task 9 Inventory/
- Short Report
- Spike Report

Technologies, Tools, and Resources used:
List of information needed by someone trying to reproduce this work
- Visual Studio 2019
- C plus plus reference (https://www.cplusplus.com/reference/)
- Zorkish game specifications

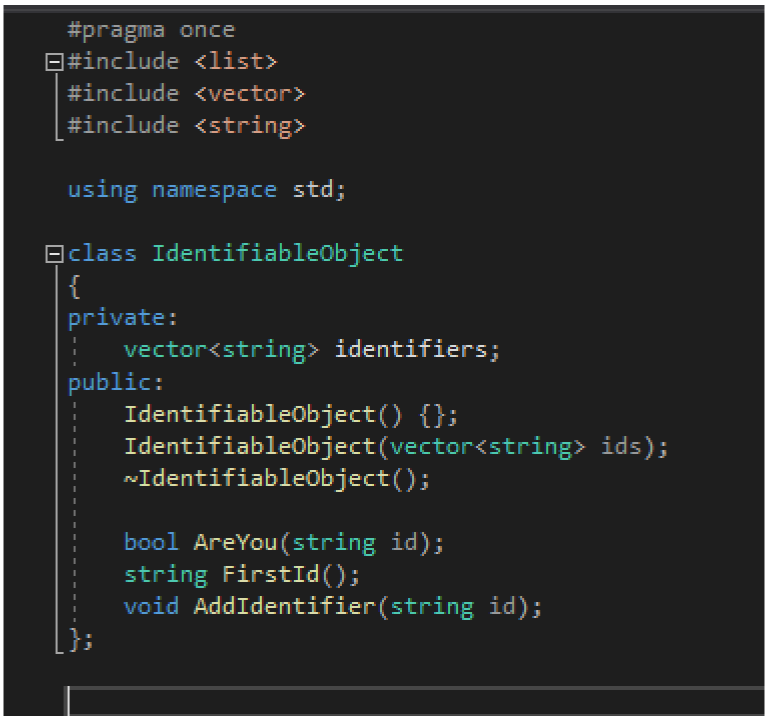
Tasks undertaken:
- Download and install Visual Studio
- Create a new C++ project
- Create identifiable object class
- Create game object class
- Create item class
- Create inventory
- Create player
- tests

What we found out:

To create the inventory system in Zorkish I created an inventory class which has a vector member variable that stores pointers to Item objects.

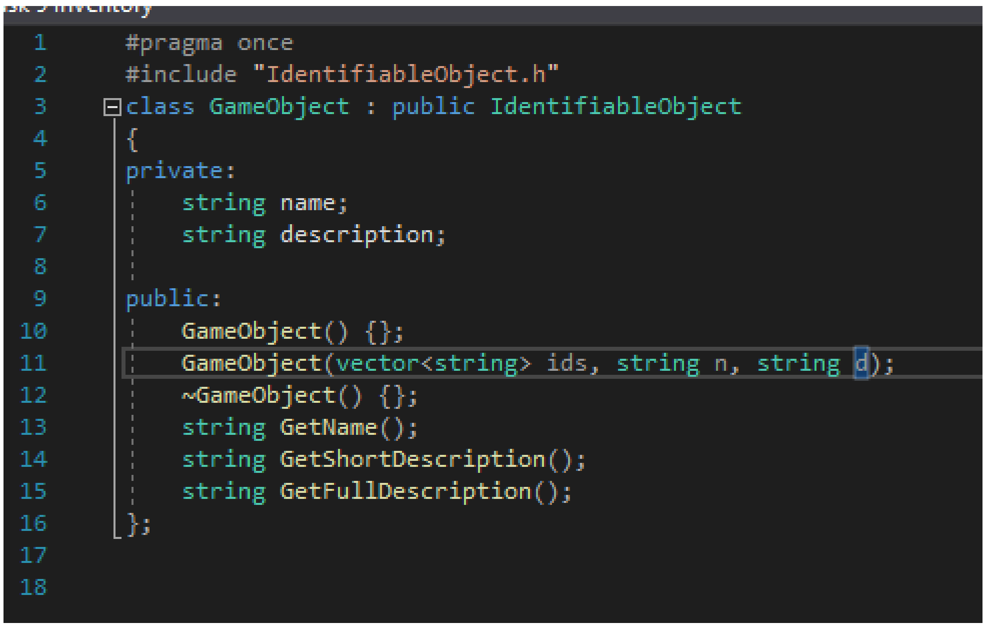
First I create the items to be stored in the inventory. I plan on having most classes used for the Zorkish game implementation being child classes of several base objects.

First there is the Identifiable object. This class has a vector of ids that we can use to identify what this object is using it's member functions. It has member functions to see if the object has an identifier, get the first identifier and also to add identifiers. This will be used for a wide range of things including commands and objects in the game.



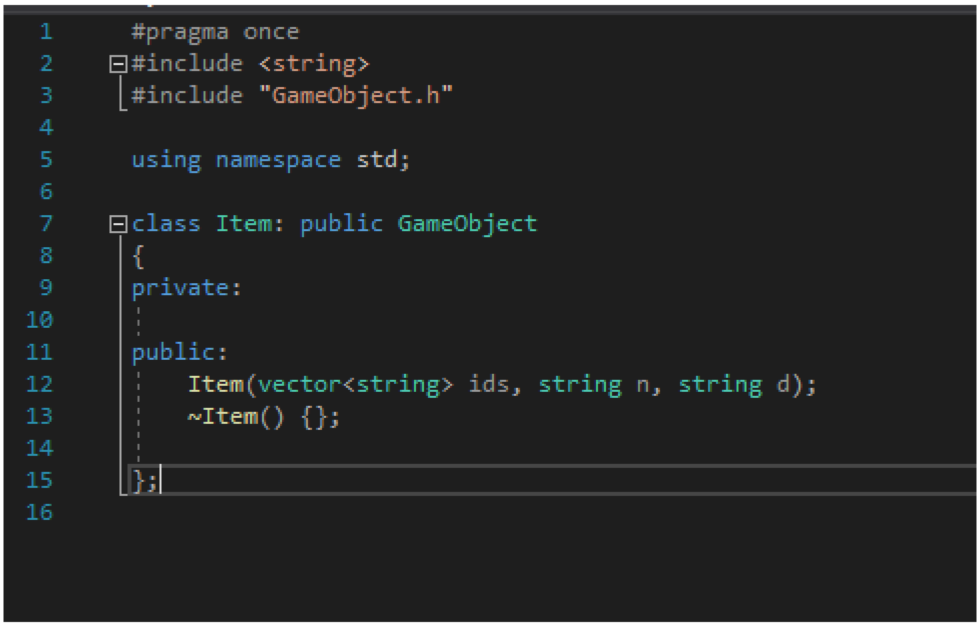
```cpp
#pragma once
#include <list>
#include <vector>
#include <string>

using namespace std;

class IdentifiableObject
{
private:
    vector<string> identifiers;
public:
    IdentifiableObject() {};
    IdentifiableObject(vector<string> ids);
    ~IdentifiableObject();

    bool AreYou(string id);
    string FirstId();
    void AddIdentifier(string id);
};
```

Next is the game object which is a child class of the identifiable object class. It adds name and description variables as well as functions to get different sized descriptions. This class is for things like items, entities, the player, locations, etc.



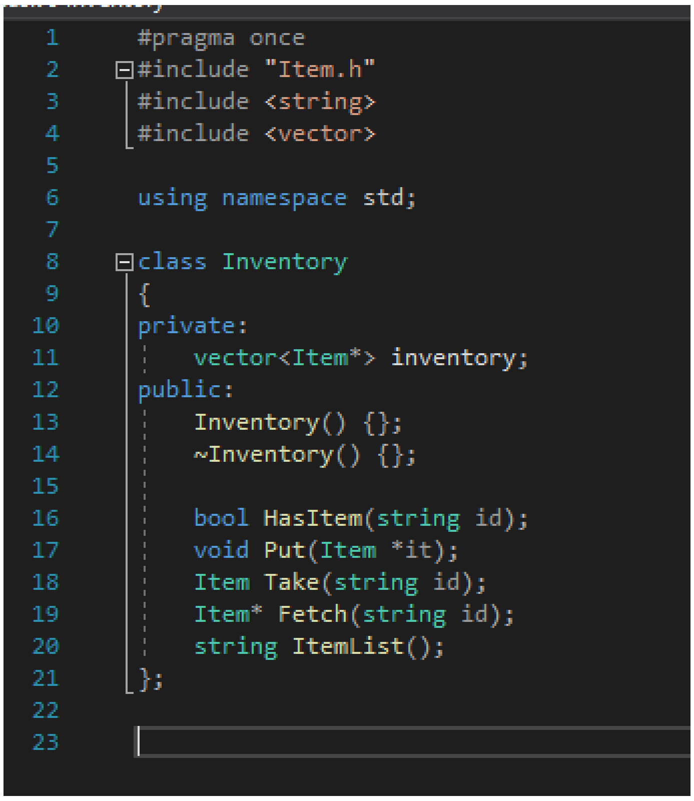
```cpp
#pragma once
#include "IdentifiableObject.h"
class GameObject : public IdentifiableObject
{
private:
    string name;
    string description;

public:
    GameObject() {};
    GameObject(vector<string> ids, string n, string d);
    ~GameObject() {};
    string GetName();
    string GetShortDescription();
    string GetFullDescription();
};
```

Now we can add items. This class is pretty simple and doesn’t change much. This is to specify the types of game objects that we want in our inventory. If we let our inventory just store regular game objects then it would be able to store things like locations which we don’t want. This also lets us make items sub classes for things like weapons, bags, etc. which can add more functionality but since they inherit from the item class they would be able to be stored in the inventory.



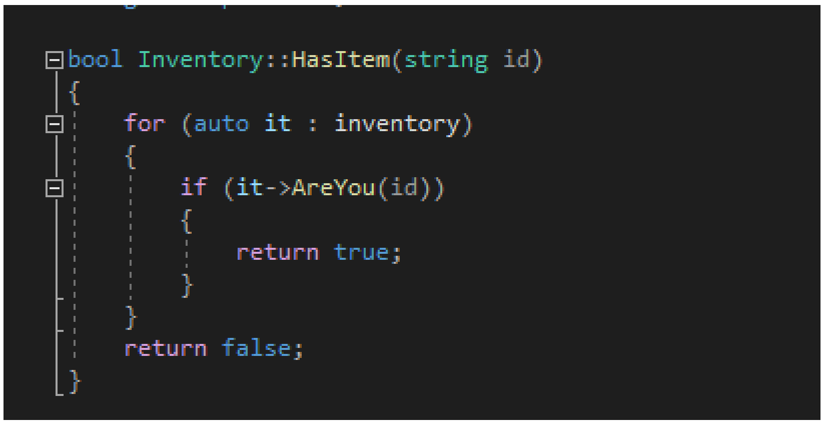
```cpp
#pragma once
#include <string>
#include "GameObject.h"

using namespace std;

class Item: public GameObject
{
private:

public:
    Item(vector<string> ids, string n, string d);
    ~Item() {};

};

```

After I added the item class I could create the inventory class. It has a vector that points to items. It also has a few member functions that manage the inventory for us.



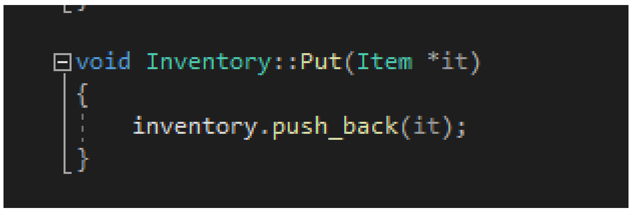
```cpp
#pragma once
#include "Item.h"
#include <string>
#include <vector>

using namespace std;

class Inventory
{
private:
    vector<Item*> inventory;
public:
    Inventory() {};
    ~Inventory() {};

    bool HasItem(string id);
    void Put(Item *it);
    Item Take(string id);
    Item* Fetch(string id);
    string ItemList();
};


```

First the has item function takes in an id and checks to see if it has an item that matches the id. This can be used for commands like looking at or taking items. We need to see if we have it first.



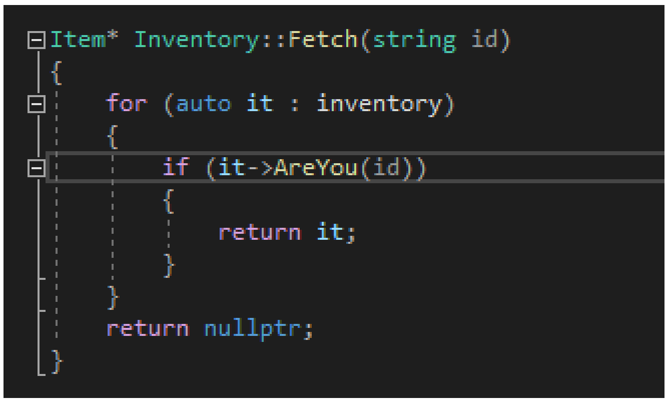
```
bool Inventory::HasItem(string id)
{
    for (auto it : inventory)
    {
        if (it->AreYou(id))
        {
            return true;
        }
    }
    return false;
}
```

Next is the put function which adds an item pointer to the vector.



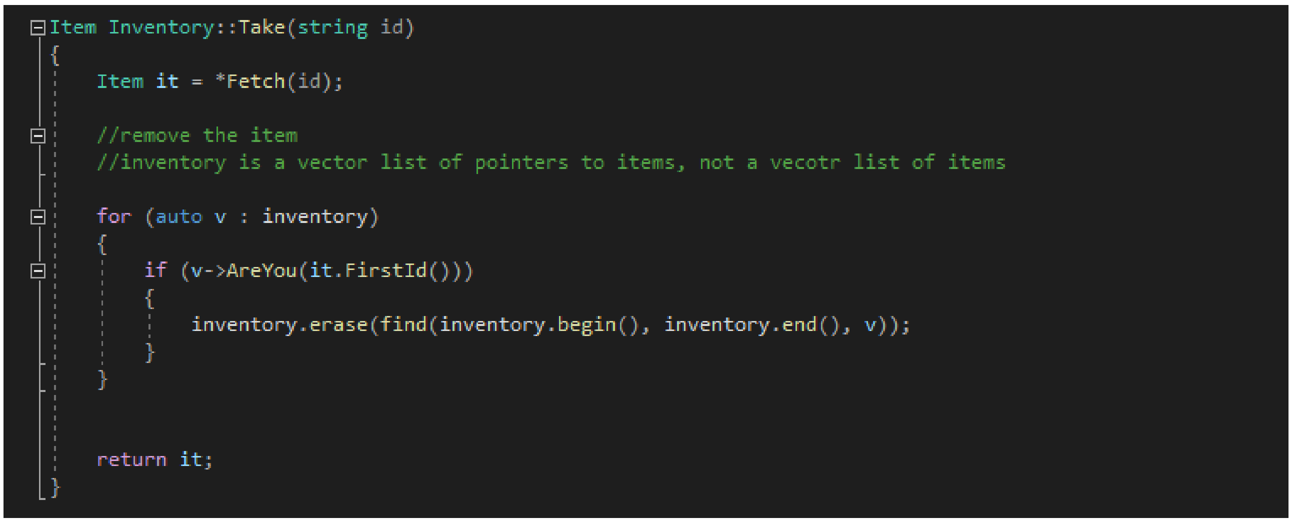
```
void Inventory::Put(Item *it)
{
    inventory.push_back(it);
}
```

The fetch function returns a pointer to an item in our inventory based on the id passed in. This could be used when a player wants to use an item or look at an item.



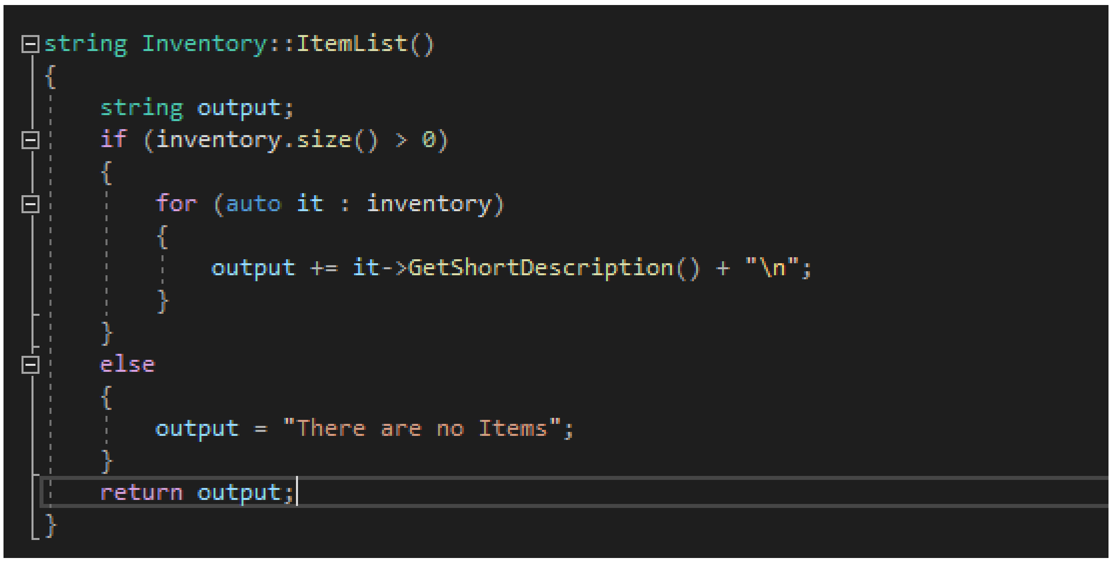
```
Item* Inventory::Fetch(string id)
{
    for (auto it : inventory)
    {
        if (it->AreYou(id))
        {
            return it;
        }
    }
    return nullptr;
}
```

The take function fetches an item and returns its value (not a pointer) while also removing it from the inventory. This could be used when a player uses a one use item or transfers an item between inventories like taking an item out of a location or putting it somewhere in one.



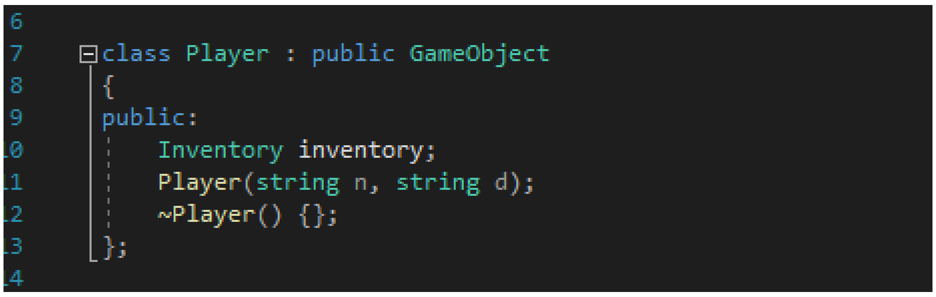
```cpp
Item Inventory::Take(string id)
{
    Item it = *Fetch(id);

    //remove the item
    //inventory is a vector list of pointers to items, not a vecotr list of items

    for (auto v : inventory)
    {
        if (v->AreYou(it.FirstId()))
        {
            inventory.erase(find(inventory.begin(), inventory.end(), v));
        }
    }


    return it;
}
```

Lastly the item list function gets every item stored in the inventory and returns a list of decryptions as a string. This can be used when a player looks at their inventory.



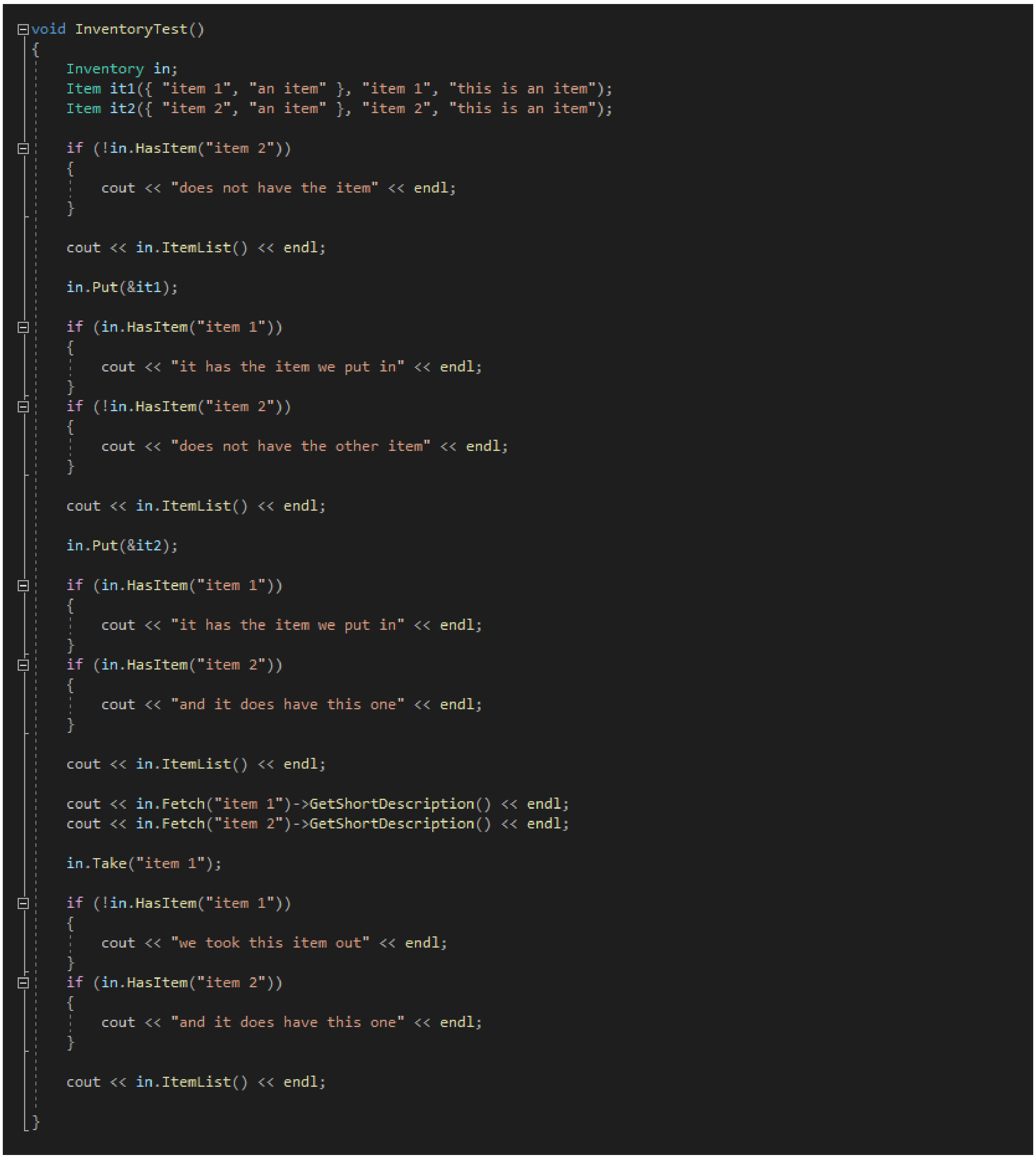
```cpp
string Inventory::ItemList()
{
    string output;
    if (inventory.size() > 0)
    {
        for (auto it : inventory)
        {
            output += it->GetShortDescription() + "\n";
        }
    }
    else
    {
        output = "There are no Items";
    }
    return output;
}
```

Now we can add a player class and give it an inventory object.

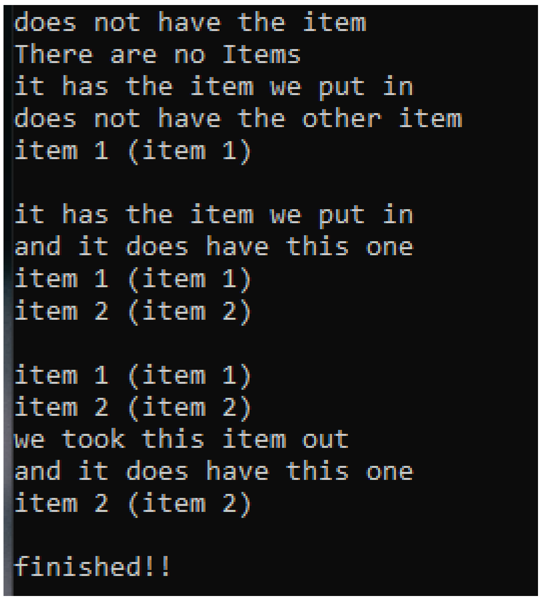


```cpp
class Player : public GameObject
{
public:
    Inventory inventory;
    Player(string n, string d);
    ~Player() {};
};
```

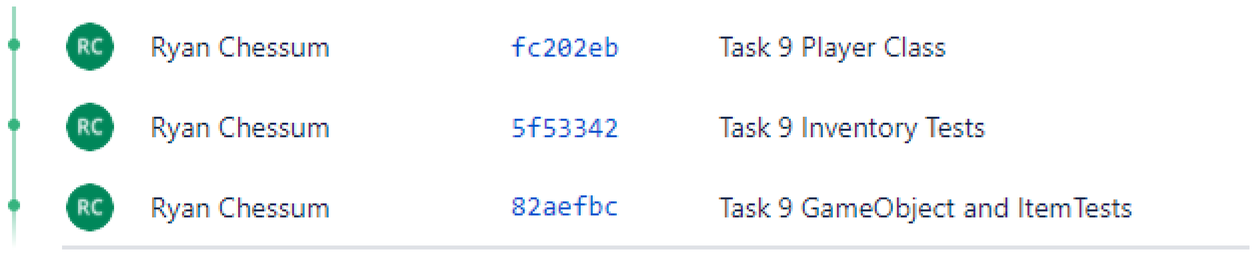
I made a few tests to see if everything worked as intended and it did.



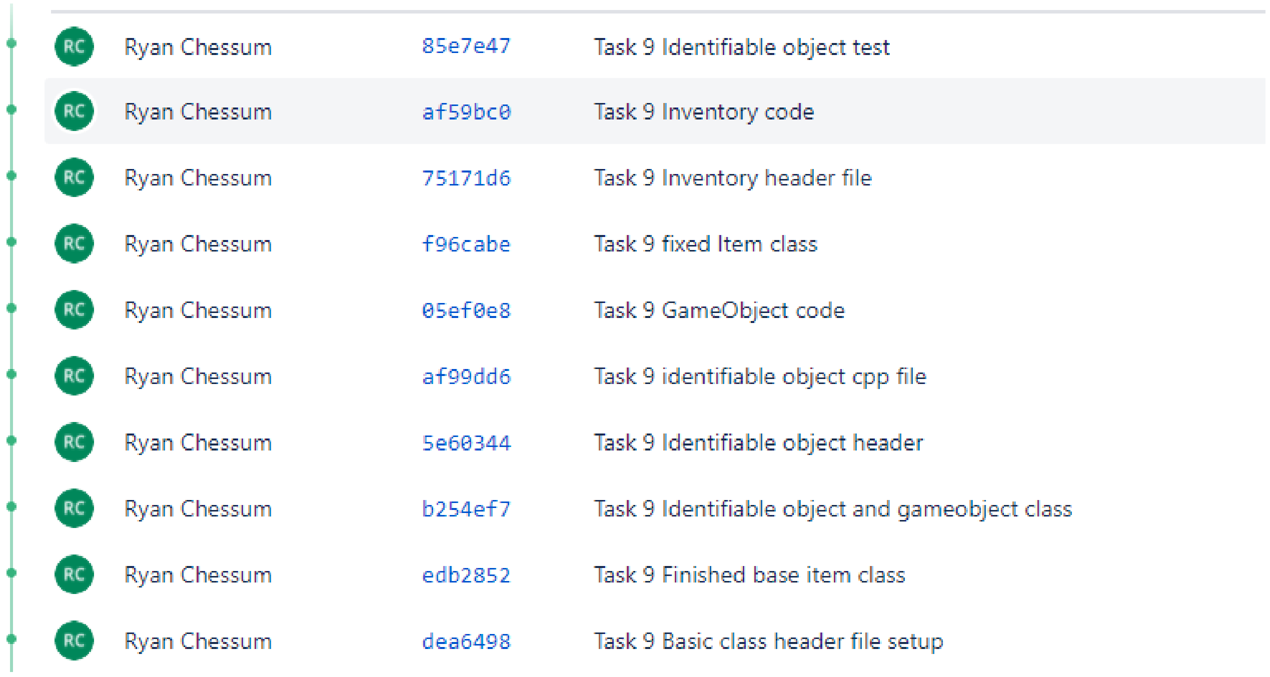
```cpp
void InventoryTest()
{
    Inventory in;
    Item it1({ "item 1", "an item" }, "item 1", "this is an item");
    Item it2({ "item 2", "an item" }, "item 2", "this is an item");

    if (!in.HasItem("item 2"))
    {
        cout << "does not have the item" << endl;
    }

    cout << in.ItemList() << endl;

    in.Put(&it1);

    if (in.HasItem("item 1"))
    {
        cout << "it has the item we put in" << endl;
    }
    if (!in.HasItem("item 2"))
    {
        cout << "does not have the other item" << endl;
    }

    cout << in.ItemList() << endl;

    in.Put(&it2);

    if (in.HasItem("item 1"))
    {
        cout << "it has the item we put in" << endl;
    }
    if (in.HasItem("item 2"))
    {
        cout << "and it does have this one" << endl;
    }

    cout << in.ItemList() << endl;

    cout << in.Fetch("item 1")->GetShortDescription() << endl;
    cout << in.Fetch("item 2")->GetShortDescription() << endl;

    in.Take("item 1");

    if (!in.HasItem("item 1"))
    {
        cout << "we took this item out" << endl;
    }
    if (in.HasItem("item 2"))
    {
        cout << "and it does have this one" << endl;
    }

    cout << in.ItemList() << endl;

}
```



```
does not have the item
There are no Items
it has the item we put in
does not have the other item
item 1 (item 1)

it has the item we put in
and it does have this one
item 1 (item 1)
item 2 (item 2)

item 1 (item 1)
item 2 (item 2)
we took this item out
and it does have this one
item 2 (item 2)

finished!!
```



RC	Ryan Chessum	fc202eb	Task 9 Player Class
RC	Ryan Chessum	5f53342	Task 9 Inventory Tests
RC	Ryan Chessum	82aefbc	Task 9 GameObject and ItemTests



RC	Ryan Chessum	85e7e47	Task 9 Identifiable object test
RC	Ryan Chessum	af59bc0	Task 9 Inventory code
RC	Ryan Chessum	75171d6	Task 9 Inventory header file
RC	Ryan Chessum	f96cabe	Task 9 fixed Item class
RC	Ryan Chessum	05ef0e8	Task 9 GameObject code
RC	Ryan Chessum	af99dd6	Task 9 identifiable object cpp file
RC	Ryan Chessum	5e60344	Task 9 Identifiable object header
RC	Ryan Chessum	b254ef7	Task 9 Identifiable object and gameobject class
RC	Ryan Chessum	edb2852	Task 9 Finished base item class
RC	Ryan Chessum	dea6498	Task 9 Basic class header file setup