Spike: Spike No.9
Title: Task 9 – Game Data Structure

Author: Khang Trinh - 102118468

Goals / Deliverables:

The goal of this spike is to help the developer learn how to figure out the data type they should be using in a given scenario (in this case, an inventory system) and how to find info about using them.

Technologies, Tools, and Resources used:

- Visual Studio 2017

Useful Links:

1. Everything one needs to know about all the available containers in C++
   https://en.cppreference.com/w/cpp/container
2. A general rule of thumb for choosing the right container
   https://embeddedartistry.com/blog/2017/08/23/choosing-the-right-stl-container-general-rules-of-thumb/
3. A flow diagram to further help with choosing the right container
   https://stackoverflow.com/questions/471432/in-which-scenario-do-i-use-a-particular-stl-container
4. More about maps - the container we'll be using in this report
   https://youtu.be/nPSDR5nZzHA

Tasks undertaken:

Step 1: Figure out what you will be doing with your data in the container

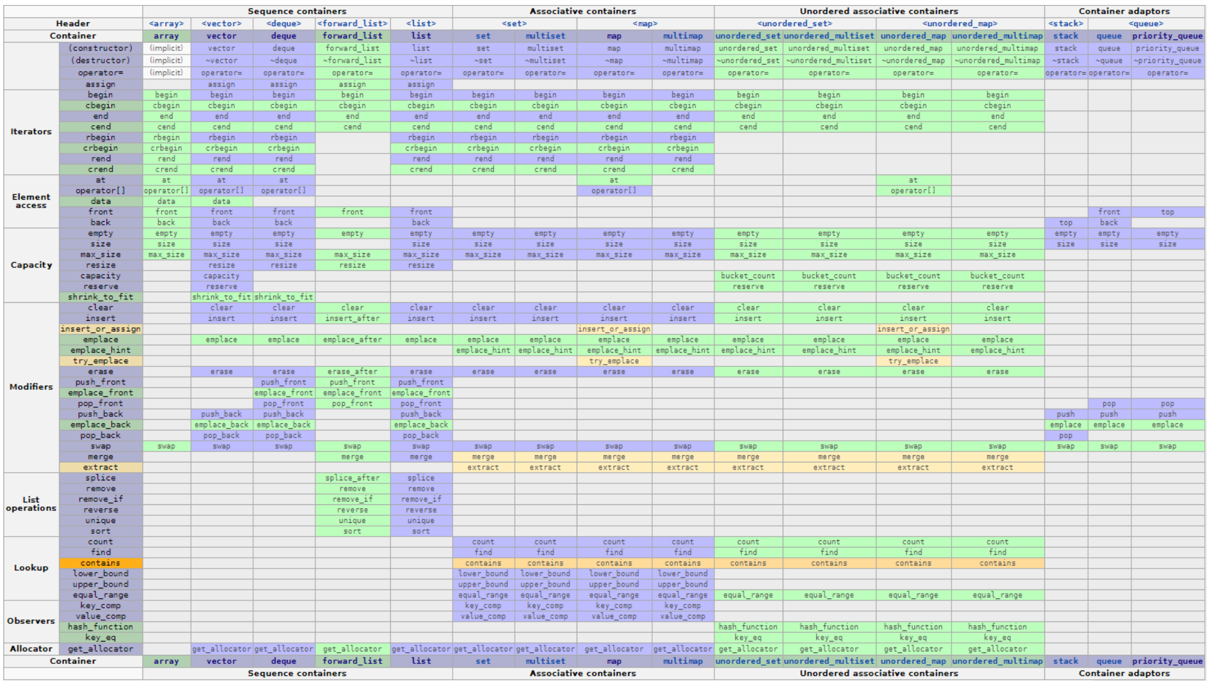
Ask yourself how your data will be used. Here are a few example questions to consider:

- How big is your container going to be? Is it fixed? Infinite? Currently unknown?
- Does your data require to be inserted in a specific slot (front/back/etc.) in the container?
- How are you going to access the data? Random? Key? Index?
- How fast/slow do you need the search for the data to be?

IMPORTANT: It's crucial that you also think about how expandable your solution can be. What works for you today might not work for you later down the line when you start to have more needs regarding how your data can be used. Can any of the conditions considered above potentially change in the future?

Step 2: Find out which container works best with what you need

One way to find out which container you need is to know all of the possible containers that could fit your needs. With this, primary source is the most reliable (refer to link 1). When you come across a new container, make note of its properties (size restriction, how it handles data insertion/removal/accessing, unique features) and consider how it could meet your needs.



		Sequence containers					Associative containers				Unordered associative containers				Container adaptors		
Header		<array>	<vector>	<deque>	<forward_list>	<list>	<set>		<map>		<unordered_set>		<unordered_map>		<stack>	<queue>	
Container		**array**	**vector**	**deque**	**forward_list**	**list**	**set**	**multiset**	**map**	**multimap**	**unordered_set**	**unordered_multiset**	**unordered_map**	**unordered_multimap**	**stack**	**queue**	**priority_queue**
	(constructor)	(implicit)	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_multimap	stack	queue	priority_queue
	(destructor)	(implicit)	~vector	~deque	~forward_list	~list	~set	~multiset	~map	~multimap	~unordered_set	~unordered_multiset	~unordered_map	~unordered_multimap	~stack	~queue	~priority_queue
	operator=	(implicit)	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=
	assign		assign	assign	assign	assign											
Iterators	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin			
	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin			
	end	end	end	end	end	end	end	end	end	end	end	end	end	end			
	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend			
	rbegin	rbegin	rbegin	rbegin		rbegin	rbegin	rbegin	rbegin	rbegin							
	crbegin	crbegin	crbegin	crbegin		crbegin	crbegin	crbegin	crbegin	crbegin							
	rend	rend	rend	rend		rend	rend	rend	rend	rend							
	crend	crend	crend	crend		crend	crend	crend	crend	crend							
Element access	at	at	at	at					at				at				
	operator[]	operator[]	operator[]	operator[]					operator[]				operator[]				
	data	data	data														
	front	front	front	front	front	front										front	top
	back	back	back	back		back									top	back	
Capacity	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty
	size	size	size	size		size	size	size	size	size	size	size	size	size	size	size	size
	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size			
	resize		resize	resize	resize	resize											
	capacity		capacity								bucket_count	bucket_count	bucket_count	bucket_count			
	reserve		reserve								reserve	reserve	reserve	reserve			
	shrink_to_fit		shrink_to_fit	shrink_to_fit													
Modifiers	clear		clear	clear	clear	clear	clear	clear	clear	clear	clear	clear	clear	clear			
	insert		insert	insert	insert_after	insert	insert	insert	insert	insert	insert	insert	insert	insert			
	insert_or_assign								insert_or_assign				insert_or_assign				
	emplace		emplace	emplace	emplace_after	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace			
	emplace_hint						emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint			
	try_emplace								try_emplace				try_emplace				
	erase		erase	erase	erase_after	erase	erase	erase	erase	erase	erase	erase	erase	erase			
	push_front			push_front	push_front	push_front											
	emplace_front			emplace_front	emplace_front	emplace_front											
	pop_front			pop_front	pop_front	pop_front										pop	pop
	push_back		push_back	push_back		push_back									push	push	push
	emplace_back		emplace_back	emplace_back		emplace_back									emplace	emplace	emplace
	pop_back		pop_back	pop_back		pop_back									pop		
	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap
	merge				merge	merge	merge	merge	merge	merge	merge	merge	merge	merge			
	extract						extract	extract	extract	extract	extract	extract	extract	extract			
List operations	splice				splice_after	splice											
	remove				remove	remove											
	remove_if				remove_if	remove_if											
	reverse				reverse	reverse											
	unique				unique	unique											
	sort				sort	sort											
Lookup	count						count	count	count	count	count	count	count	count			
	find						find	find	find	find	find	find	find	find			
	contains						contains	contains	contains	contains	contains	contains	contains	contains			
	lower_bound						lower_bound	lower_bound	lower_bound	lower_bound							
	upper_bound						upper_bound	upper_bound	upper_bound	upper_bound							
	equal_range						equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range			
Observers	key_comp						key_comp	key_comp	key_comp	key_comp							
	value_comp						value_comp	value_comp	value_comp	value_comp							
	hash_function										hash_function	hash_function	hash_function	hash_function			
	key_eq										key_eq	key_eq	key_eq	key_eq			
Allocator	get_allocator		get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator			
Container		**array**	**vector**	**deque**	**forward_list**	**list**	**set**	**multiset**	**map**	**multimap**	**unordered_set**	**unordered_multiset**	**unordered_map**	**unordered_multimap**	**stack**	**queue**	**priority_queue**
		Sequence containers					Associative containers				Unordered associative containers				Container adaptors		

Fig 1. Table showing all the features available for each type of container on cppreference

Another way to find out which container you need is to ask what others use/would use if they were in your situation. You might get mixed answers, but at least that could give you some pointers in where you could be looking. For example, if people you ask tell you that using a vector or a map would be the best way to create an inventory system, then those two should be your top priority to look into.

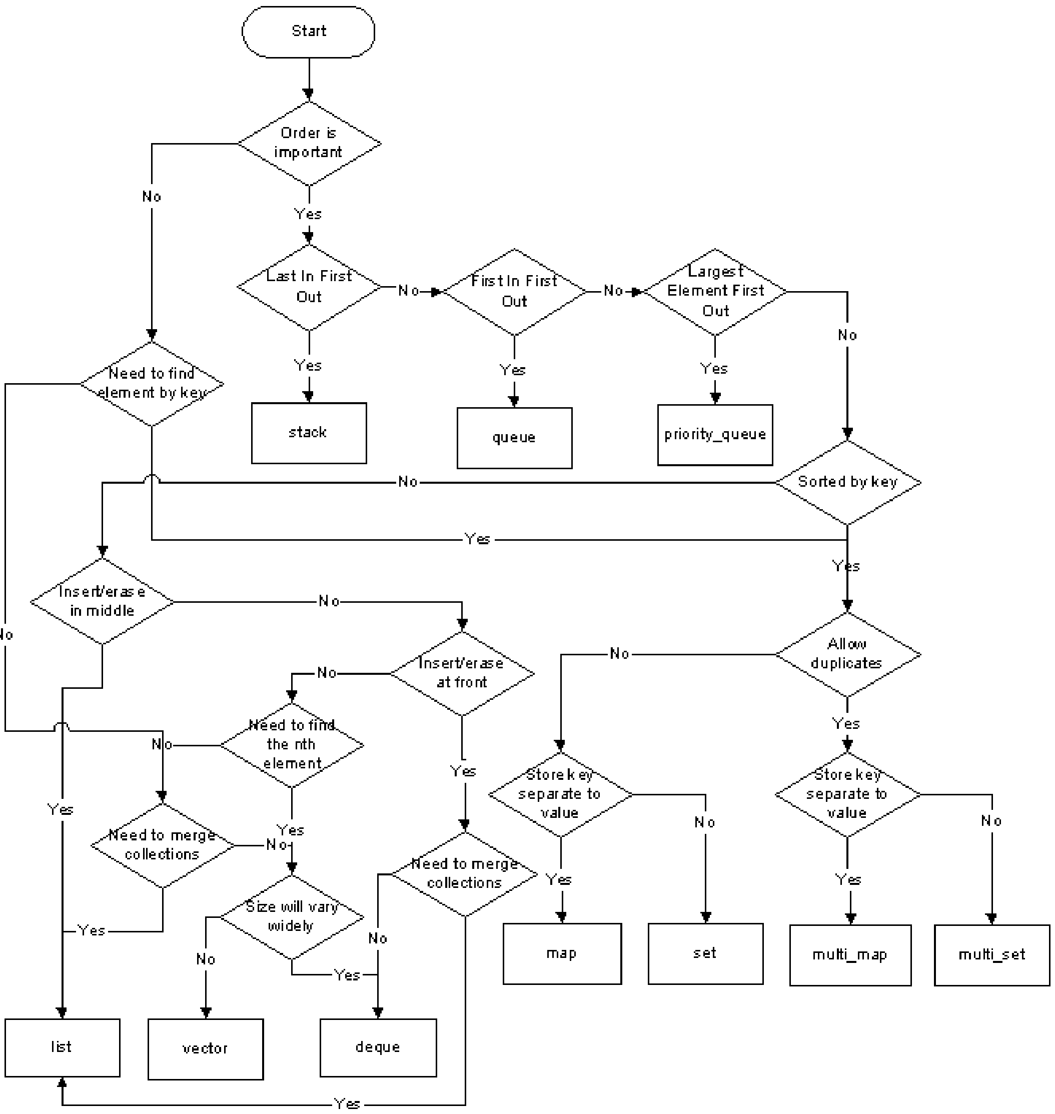
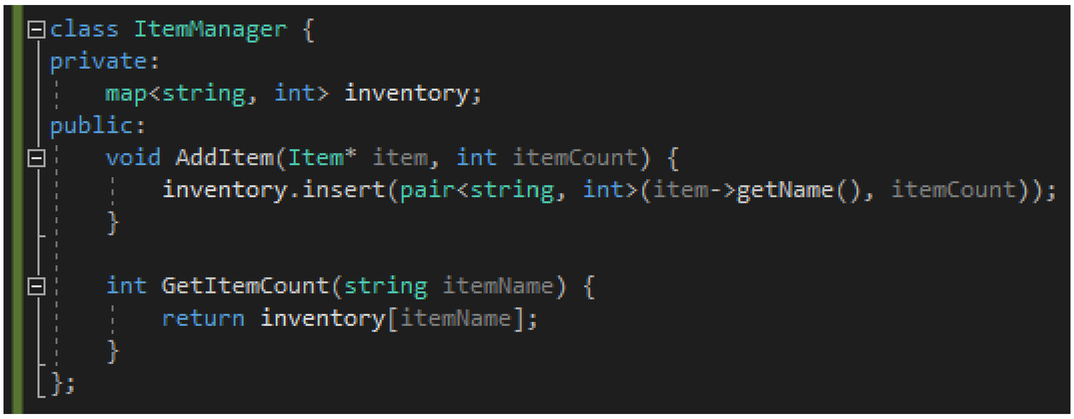


Fig 2. A flow chart that could help you decide which container to use

Step 3: Learn and implement!

Say you've chosen to use a map for your inventory, but you have no idea how to use it. All containers have a way to declare, add/remove/modify things. Google your way through while keeping this in mind until you've learned how to implement the container of your choice and make use of its strong points.



```
class ItemManager {
private:
    map<string, int> inventory;
public:
    void AddItem(Item* item, int itemCount) {
        inventory.insert(pair<string, int>(item->getName(), itemCount));
    }

    int GetItemCount(string itemName) {
        return inventory[itemName];
    }
};
```

Fig 3. An example of an inventory system