


1단계_10. UCL #1 Array와 Set

📅 수강일	@2023/04/03
➦ 이름	 <u>전영재</u>
🔍 멘토	Min-Kang Song, 현웅 최
👥 멘티	
⚡ 작성 상태	Not started
📁 단계	
☑ 강의 시청 여부	<input checked="" type="checkbox"/>
☑ 이수 여부	<input type="checkbox"/>

Contents



[강의 제목](#)

[0n. ~~~ 강의 과제](#)

강의 제목



강의 목표

- 언리얼 대표 컨테이너 라이브러리 TArray, TSet의 내부 구조 이해
- 각 컨테이너 라이브러리의 장단점을 파악하고, 알맞게 활용하는 방법의 학습

언리얼 컨테이너 라이브러리

언리얼엔진이 자체 제작해 제공하는 자료구조 라이브러리
UCL(Unreal Container Library)이라고도 한다.
유용한 라이브러리는 TArray, TMap, TSet 3가지가 있다

차이점

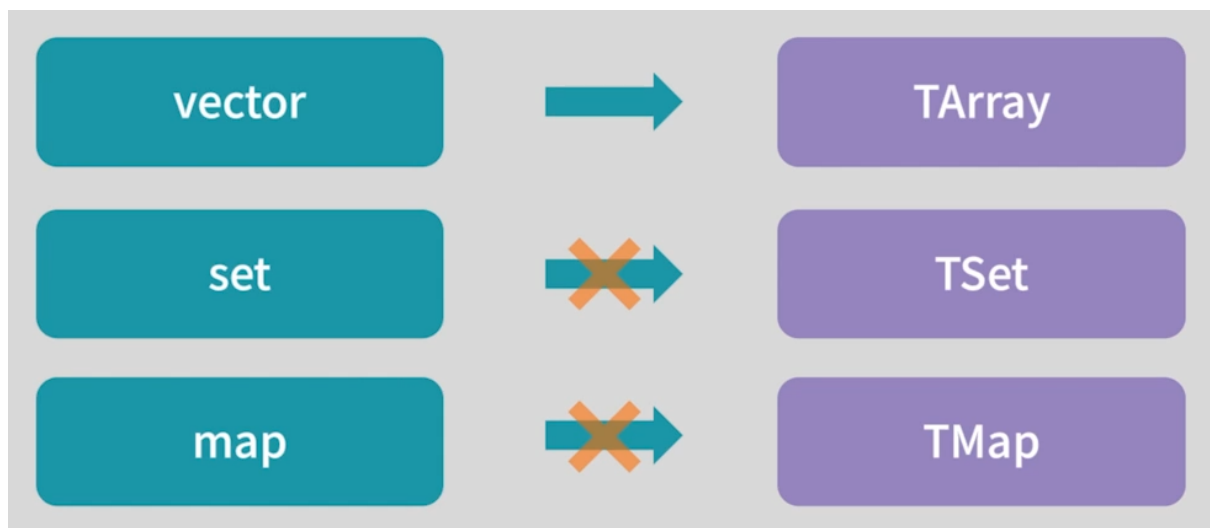
C++STL은 범용적이고 표준이기에 호환성이 높다.

그러나 너무 많은 기능이 있기에 컴파일 시간이 오래걸린다.

UCL은 언리얼에 특화되어 있어 가볍고 게임제작에 최적화 되어있으며 안정적이다. 그러니
이거쓰자!

Set와 Map은 이름과 용도는 유사하지만, 내부적으로는 다르게 구현되어 있다.

- 두 라이브러리의 이름과 용도는 유사하지만, 내부적으로 다르게 구현되어 있음.
 - TArray : 오브젝트를 순서대로 담아 효율적으로 관리하는 용도로 사용
 - TSet : 중복되지 않는 요소로 구성된 집합을 만드는 용도로 사용
 - TMap : 키, 밸류 조합의 레코드를 관리하는 용도로 사용



TArray 개요

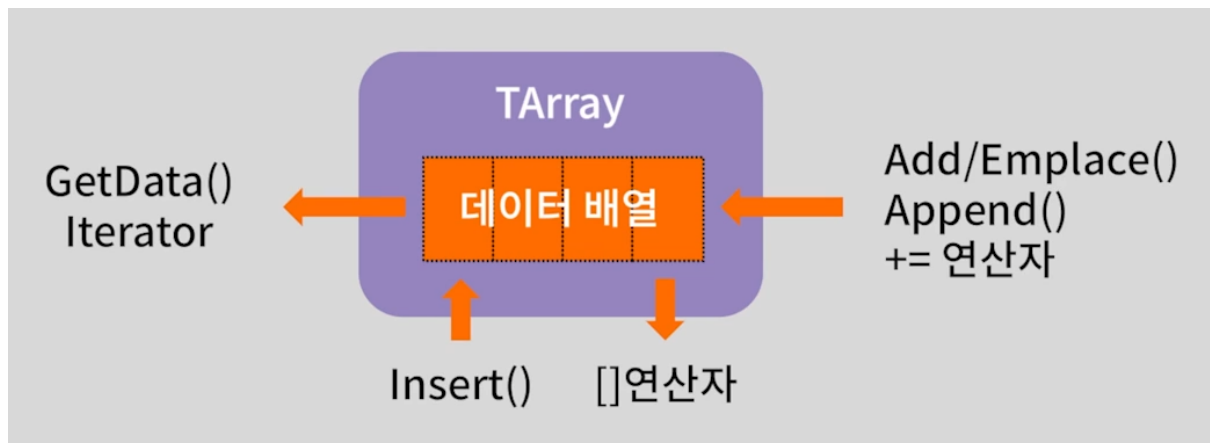
맘대로 크기를 변경할 수 있는 가변배열(Dynamic Array) 자료구조

STL의 Vector와 동작원리가 유사함.

장단점과 언제 쓸지?

- 게임 제작에서는 가변 배열 자료구조를 효과적으로 활용하는 것이 좋음.
 - 데이터가 순차적으로 모여있기 때문에 메모리를 효과적으로 사용할 수 있고 캐시 효율이 높다.
 - 컴퓨터 사양이 좋아지면서, 캐시 지역성(Locality)으로 인한 성능 향상은 굉장히 중요해짐.
 - 임의 데이터의 접근이 빠르고, 고속으로 요소를 순회하는 것이 가능.
- 가변 배열의 단점
 - 맨 끝에 데이터를 추가하는 것은 가볍지만, 중간에 요소를 추가하거나 삭제하는 작업은 비용이 큼
- 데이터가 많아질 수록 검색, 삭제, 수정 작업이 느려지기 때문에, 많은 수의 데이터에서 검색 작업이 빈번하게 일어난다면 TArray대신 TSet을 사용하는 것이 좋음.

내부구조



insert와 remove는 비용이 많이들지만, 데이터가 같은유형이기에 []로 중간선택은 빠르게 작동한다.

가장 간단한 컨테이너이면서도 빠르기에 가장 자주 쓰인다.

Document 읽어보기(처음부터 끝까지 모든 항목을 다 읽는다.)

allocator는 자주 안쓴다. 대부분 기본값 쓴다.

특수하게 메모리 관리가 필요할때 직접 작성해서 사용함.

TArray가 '값유형' 이라는 말은 동적할당을 하지 않는다는 것임. 즉, new나 delete를 사용하지 않는게 좋다.

정의 방법

```
TArray<int32> Array;

//채우기는 여러 방식으로 가능하다.
Array.Init(10, 5);
//Array == [10, 10, 10, 10, 10]
```

Add 또는 **Push** 는 추가할 데이터를 밖에서 생성하고 TArray안에 복사해서 붙여넣는거고,

Emplace 는 TArray안에서 즉시 생성하는 것임. 그래서 좀 더 효율적이다.

그러니 사소한건 Add, 그외에는 Emplace를 써라. Add는 가독성이 좋아서 사용된다.

절대 Emplace가 Add보다 효율이 떨어지는 일은 없다.

Append 는 한번에 여러 엘리먼트를 넣을때 사용한다.

AddUnique도 있는데 별로 효율적이지 않음. 차라리 TSet써라. 궁금하면 document 찾아보기.

Insert = 해당 인덱스에 집어넣기

SetNum = 배열의 크기를 인위적으로 조정가능

반복처리

일반적으로 for문 중 range 기능을 사용한다.

```
for (auto& Str : StrArr) //이게 range
```

sort도 종류가 여러개다.

쿼리 (내부구조 그런건듯?)

Num함수로 배열에 엘리먼트가 몇개인지 확인가능

GetData로 엘리먼트에 대한 포인터를 반환할 수 있다.

...

연산자

할당연산자로 값을 넣을수있는데 1, 이때 값을 복사해서 넣는다는게 중요하다. like Add함수처럼

```
auto ValArr2 = ValArr1;  
//이러면 ValArr1이 복사되서 2에게 들어간다는것
```

아예 이주하고 싶다면 MoveTemp() 함수 사용하기

좀 사소할 수도 있는데, 오퍼레이터 중 '=='같은 경우는 엘리먼트 수와 순서 모두 같아야 true 다.

//참고로 이때 대소문자는 구분하지 않는다.

힙

이진힙(BinaryHeap)구조를 알고 봐야 이해될거다.

슬랙

메모리가 확보되어야, 즉 정보를 넣을 자리가 있어야 넣을수있는건데 배열의 크기 변경을 하나 하나씩 하면 메모리 오버헤드가 생긴다. 그렇기에 요청한것보다 좀더 넉넉하게 데이터를 잡는다.

즉, 여유분(slack)을 잡는다는것. Vector의 큰 특징

그런데 TArray는 이런 slack자체도 깔끔하게 정리해주는 기능이 있다더라. 그런 방법을 추가로 설명하고 있다.

TArray 예제 //예제 코드자체를 쓸필요 없을것같다.

int32타입의 TArray를 만들고, 채우고(Add로), 짝수부분을 제거하고, 오퍼레이터로 다시 채워넣는 예제였다.

짝수부분제거 : RemoveAll([람다]() {조건}); 을 사용해서 사용하더라.

참고로 이 배열의 내용을 보기위해서는 언리얼용으로 최적화된거라 바로 보이지는 않고 DebugGame Editor로 빌드를 바꾼다음에 찾아봐야한다.

그후엔 별개의 Compare용 Array를 만들고, 여기다가 int32짜리 배열(Tarray아님)을 만들어 초기화하지 않고 넣은뒤(AddUninitialized로 초기화하고, Memcpy로 채운다.) 앞서 만든 배열과 같은지 ==로 비교해봤다.

TArray를 사용하는 알고리즘 라이브러리?

대표적으로 합을 구하는 알고리즘.

#include "Algo/Accumulate.h" 를 추가해 accumulate를 사용해보자.

보통 배열 안의 값들의 합을 알고싶으면 for문 써서 sum을 구할텐데, 이렇게 복잡해진다면

Algo::Accumulate(Array, 0) //0은 시작 인덱스를 뜻한다.

를 사용하면 한줄에 바로 합이 되더라.

TSet의 구조와 활용

TSet의 특징

STL은 내부적으로 이진트리라 정렬을 지원함. 그런데 정렬때문에 효율적이진 않음(삭제하면 균형맞추려고 재구축하는것처럼). 그리고 트리라서 모든자료를 순회하는데 적합하지 않음.

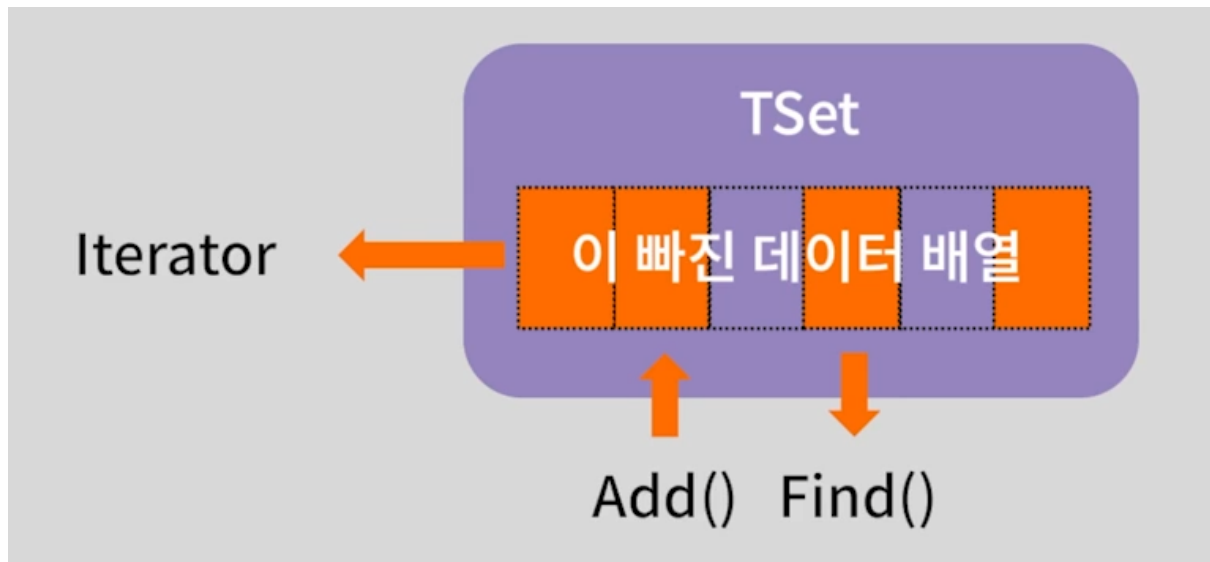
그런데 언리얼 TSet는 해시테이블 형태로 키데이터가 구축되어서 빠르게 검색 가능함.

그리고 동적배열 형태로 데이터가 모여있다. 그리고 데이터 삭제해도 재구축이 일어나지 않음. 그렇기에 Tset의 배열에는 비어있는 부분이 생길수도 있다 (TArray는 알아서 땡겨주는 것과 다르다.)

오히려 STL의 unordered_set과 비슷함. 물론 동일하진 않음.

즉, TSet은 중복없는(해시 쓰니까!) 데이터 집합 구축에 유용하다.

TSet 내부구조



동적 가변배열 + 중간중간 데이터 빠져있음 + 해시테이블

Document 읽어보기

map과 중요한 차이점? ⇒ 키-밸류 쌍이 아니라, 데이터값(밸류) 자체를 키로 사용한다. //즉, 밸류를 키로쓰는 해시테이블이라고.

엘리먼트 추가, 검색, 제거가 진짜 매우 빠르다. 단, 중복키는 지원하지 않는다.

순서가 중요치 않은 상황에 고유 엘리먼트를 저장하는데 사용하는 고속 컨테이너 클래스다.

데이터 저장을 위한 커스텀 메모리 얼로케이터도 만들수 있다는데, 너무 고급내용이라 스킵

세트 생성 및 채우기

`TSet<FString> FruitSet;` //단순하게 타입을 지정해주는 것만으로 생성가능

TArray와 유사하게, 선언만 한다면 비어있는 자료구조가 생성된다.

데이터를 넣으면 GetTypeHash를 통해서 해싱을 진행하는데, FString, int32 같이 기본적으로 제공하는 타입은 그런 해싱표가 준비되어있다. 그러나 우리가 만든 클래스를 넣으려면 직접 해싱표를 만들어줘야 된다.

Add를 사용해 추가

Emplace로도 가능

Append로 다른 Set와 병합가능 //마치 집합끼리의 연산처럼

UPROPERTY TSet편집

딱히 set의 특징은 아니다..

이터레이션

for문써서 쉽게 돌릴수있다.

읽기전용, 읽고쓰기 이터레이터 두가지 이터레이터가 있다.

const 붙이면 읽기전용으로 불러올수있다.

쿼리

Num으로 엘리먼트 몇개인지 알수있다.

Contains로 자료가 있는지 확인이 가능하다,

→ TArray와 다르게 해시테이블로 되어있어서 찾는데 훨씬 빠르다.

내부적으로 `FSetElementId` 구조체로 저장되는데, 이걸 잘 안다면 `FSetElementId` 를 직접 지정해서 찾을 수도 있다.

Find : 세트에 키 포함여부를 확인할때 contains를 쓰고 operator[]를 써야되서 사실 2가지 과정을 거쳐야됨. 근데 이러면 비효율적이니 Find함수를 쓰는게 좋다더라.

TSet도 동적배열 구조라서 TArray로 반환할 수 있다. TArray는 빈틈이 없기때문에 TSet의 빈틈이 사라진 모습을 얻을수있다.

제거

내부순서가 안정적이지 않다한 것처럼, Remove함수에 인덱스를 붙여 제거하기 힘들다.

ex) `FruitSet.Remove(3);` //3번 인덱스에 머가 들어있을지, 솔직히 알기어려움. 그래서 잘못 지울수도있다.

소팅

정렬인데.. 이런게 있다더라 정도만?

연산자

복사하면 사본을 갖는다. TArray랑 비슷한듯

슬랙

여유분 메모리를 말하는건데, TSet에 중간중간 빈틈이 있다고 했던것이 이것에 해당한다.
그래서 앞,뒤,중간 어디든 슬랙이 있을수 있다. 그리고 그값은 invalid로 차있더라.

Shrink함수를 쓰면 뒤쪽의 slack만 제거된다.

DefaultKeyFuncs

커스텀 클래스로 TSet만드려면, '==오퍼레이터'랑 '해당 타입에 대한 GetTypeHash함수' 두가지를 구현해야한다.

다음 강좌에서 set과map에서 직접 함수를 사용하면서, 어떻게 선언해야 하는지 보여주겠다.

그런데, 이걸 오버로드 하지 않을거면 document에 나온 **DefaultKeyFuncs**를 보면된다.
그런데 어려운내용이라 다루진 않을거다.

기타

CountBytes및 GetAllocatedSize로 메모리 양 측정 가능하다.

TSet 실습

앞선 TArray실습에 붙여서 만들기

for문써서 넣어주고, remove로 일일이 제거해주고, 새로 값을 채워준다.

그런데 TSet는 해시를 쓰기때문에 13579246810이 나오지 않고 무작위로 섞여서 나온다.
(사실 무작위는 아니고 어려운 연산순서가 있긴함. 근데 실전에선 그거 계산못하니 그냥 무작위라고 생각하자.)

TArray와 TSet의 시간복잡도 비교

자료구조의 시간 복잡도 비교

- 각 자료구조의 시간복잡도(Time Complexity)

	TArray	TSet
접근	$O(1)$	$O(1)$
검색	$O(N)$	$O(1)$
삽입	$O(N)$	$O(1)$
삭제	$O(N)$	$O(1)$

빈틈없는 메모리
가장 높은 접근성
가장 높은 순회성능

빠른 중복 감지

Summary

- TArray, TSet의 내부구조와 특징
- DebugGame Editor 빌드를 사용해 메모리 정보를 확인하는 방법

0n. ~~~ 강의 과제

Q1.

? Q2.

? Q3.



Reference