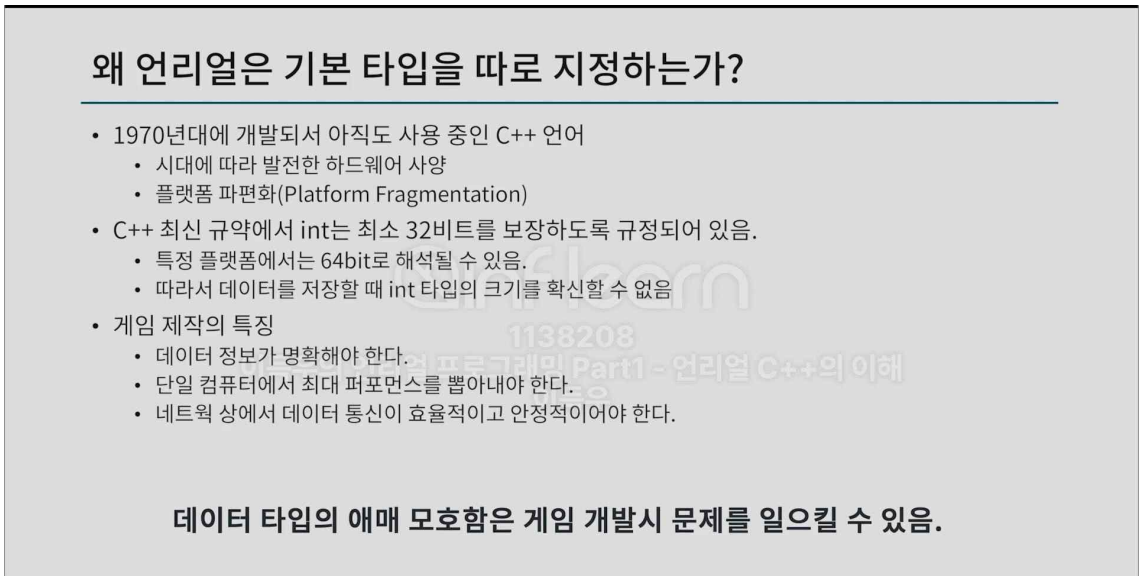
언리얼 프로그래밍 Part1-3

제목:기본 타입과 문자열

**강의 내용 : 언리얼 c++ 기본 타입과 문자열 처리

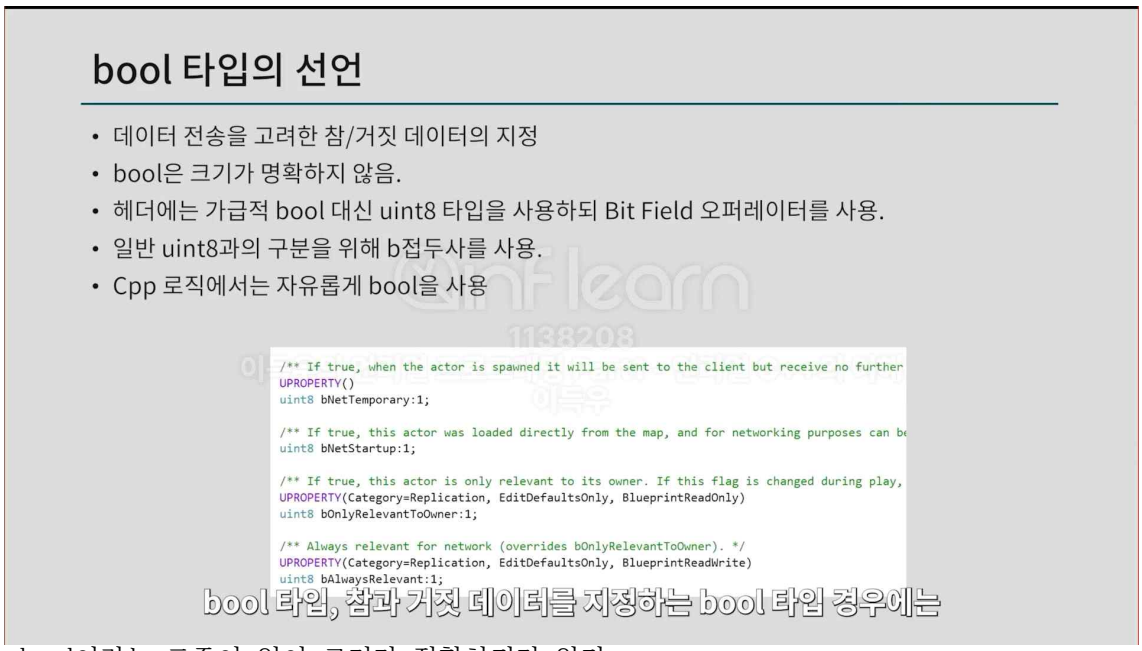
** 강의 목표
- 언리얼 환경에서 알아두어야 할 기본 타입과 고려할 점
- 캐릭터 인코딩 시스템에 대한 이해
- 언리얼 c++이 제공하는 다양한 문자열 처리 방법과 내부 구성의 이해

** 언리얼 c++ 기본타입



- 언리얼은 c++ 기본 타입을 사용하지 않는다.
- 예전에는 애플, 플레이스테이션등 기기마다 c++이 각기 다르게 사용되어 '플랫폼 파편화'가 일어났다.
- c++에 규약이 생기며 플랫폼 파편화를 막을려고 노력함.
- 게임 제작에는 다른 프로그램과 다르게 단일 컴퓨터에서 퍼포먼스를 최대한 뽑아내야 한다.
- 퍼포먼스를 높일려면 캐시 히트율을 높여야 하는데 그럴려면 데이터 정보가 캐시크기에 맞춰서 잘 정렬되어야 한다. 그런데 애매모호한 데이터 타입은 따라서 게임 개발에 문제가 생김.
- 멀티플레이에어서도 데이터 타입이 애매모호하면 성능저하와 안전성에 문제를 일으킨다.
- 따라서 언리얼 엔진도 이러한 문제를 인식하여 기본타입의 int가 아닌 자체적으로 지정한 int32를 사용한다.

**bool타입의 선언



-bool이라는 표준이 없어 크기가 정확하지가 않다.

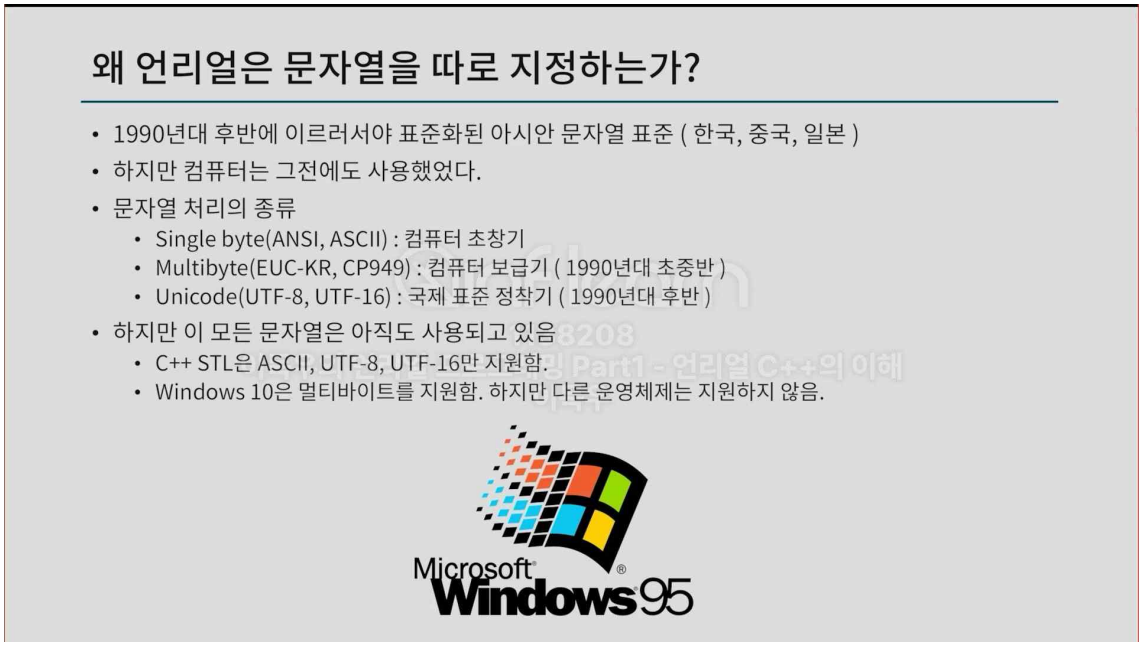
-네트워크를 전송하거나 디스크에 저장시 어떻게 저장할지 명확하지가 않다.

-헤더파일에 따르면 byte정보를 따르기로 되어있지만 byte를 사용하기엔 정보가 너무 커서 Bit Field를 사용하여 1비트로 줄여 저장한다. 또한 uint8과 구분하기 위해서 앞에 접두가 b를 붙인다.

-하지만 이것은 데이터를 저장하기 위한 고려사항이며 cpp파일 로직에서 bool타입을 사용할땐 bit flag를 명시하거나 uint사용을 하지 않고 자유롭게 bool를 사용하여도 된다.

캐릭터 인코딩

**왜 언리얼은 문자열을 따로 지정하는가?



- 언리얼은 문자열 지정을 위해 TCHAR 타입을 사용하도록 되어 있다.
- 영어는 1바이트 표현이 가능하지만 동아시아 국가의 언어들은 1바이트로 표현하기에 무리가 있음.
- 윈도우 95의 경우 유니코드가 제공되기전에 이미 만들어져 다양한 국가가 사용했는데 cp949라는 체계를 만들어 한글을 처리하였다. ==> 멀티바이트 문자열로써 1990년대 초중반에 이미 만들어짐.
- 95 이후의 운영체제는 유니코드를 지원하게 됨.
- 문제는 멀티바이트 문자열 체계가 아직 윈도우즈 운영체제에 널리 쓰이게됨
- 즉 우리가 처리해야할 문자체계가 3가지인데 언리얼엔진에서는 이를 위해 TCHAR라는 문자열 고유한 처리 방식을 제공하고 있다.
- 캐릭터 인코딩 관련 공식 링크: https://dev.epicgames.com/documentation/ko-kr/unreal-engine/character-encoding-in-unreal-engine?application_version=5.1
- 언리얼은 내부적으로 모두 UTF-16 즉 한문자당 2바이트를 사용한다.

*UE4 에 로드되는 텍스트 파일
-게임에서 대사들을 읽어들이는것과 같은 작업을 할 때는 UTF-16으로 받아들인다.

*언리얼에서 사용되는 텍스트 파일용 추천 인코딩
-유니코드를 사용해야 한다.
-언리얼에서 사용하는 설정파일들도 다 UTF-16으로 통일해야 한다.
-소스코드의 경우에는 가급적 많이 사용되는 UTF-8을 써라

*동아시아 인코딩 고유의 C++ 소스 코드에 대해서
-UTF-8과 디폴트 인코딩 모두 문제를 야기할 수 있어서 주의를 해야 한다. 하지만 이것을 정확히 알고 있다면 사용하여도 무방하다. 잘 모르면 쓰지 말자.(컴파일 에러가 났는데 왜 났는지 알수가 없을 수도 있다.)

-UTF-8로 소스코드를 저장할때도 BOM과 같이 3바이트 헤더가 들어가서 어떤 UTF-8 문서인지 지정하는 헤더가 있는데 윈도우즈 같은 경우 헤더를 넣지만, 리눅스 같은 경우 읽어 들이질 않고 안 쓰고 자체적으로 해석한다. 이런 부분에서 UTF-8을 사용할 때 문제가 야기될 수 있다.

-정리
1.전체적으로 언리얼 엔진에서 스트링을 관리할 때 UTF-16을 사용한다.
2. 소스코드에 꼭 한글을 사용한다면 UTF-8 방식을 저장한다. 하지만 여러 가지 문제가 발생할 수 있기 때문에 사전에 잘 감안하여 처리해야 한다.

TCHAR와 FString

- MyGameInstance.cpp

```
void UMyGameInstance::Init()
{
	Super::Init();
	TCHAR LogCharArray[] = TEXT("Hello Unreal");
	UE_LOG(LogTemp, Log, LogCharArray);
}
```

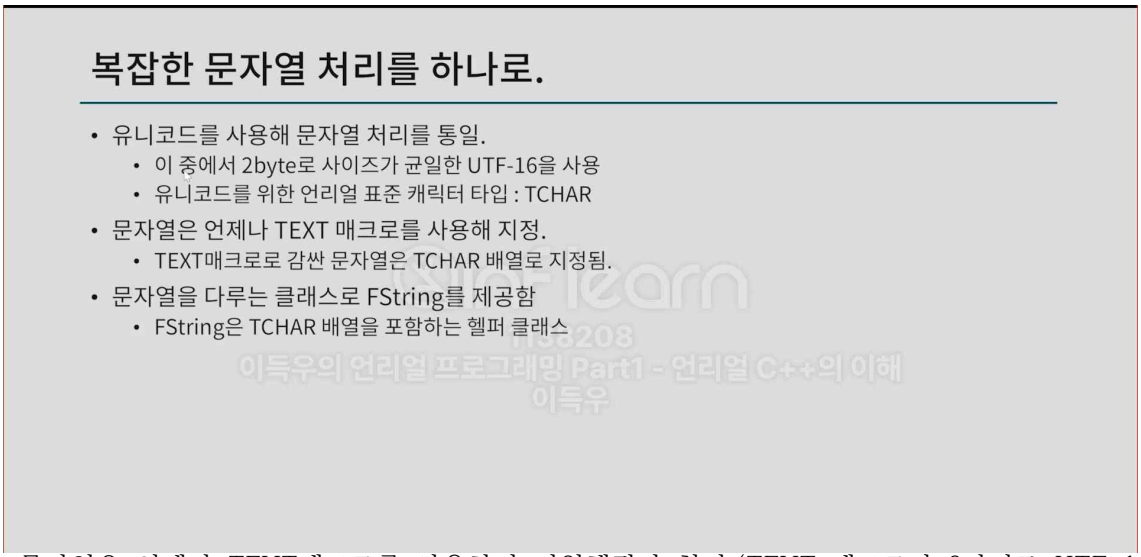
-만약 문자열을 다양하게 조작하고 싶다면 TCHAR이 아닌 언리얼엔진에서 제공하는 FString 클래스를 사용해야 한다.

-FString을 쓸 때 매크로에서 세 번째 구문에는 항상 배열만 들어가기 때문에 FString을 쓸 수가 없고 %s에 대응될 때는 TCHAR의 포인터 어레이를 반환해 줘야 되는데 FString을 그대로 쓰면, TCHAR포인터가 반환이 되지 않는다. 따라서 포인터 연산자를 지정해줘야 한다.

- MyGameInstance.cpp

```
void UMyGameInstance::Init()
{
	Super::Init();
	TCHAR LogCharArray[] = TEXT("Hello Unreal");
	UE_LOG(LogTemp, Log, LogCharArray);

	FString LogCharString = LogCharArray;
	UE_LOG(LogTemp, Log, TEXT("%s"), LogCharArray);
}
```

**복잡한 문자열 처리를 하나로.



-문자열은 언제나 TEXT매크로를 사용하여 지원해줘야 한다.(TEXT 매크로가 2바이트 UTF-16으로 내부에서 반환이 된다.)

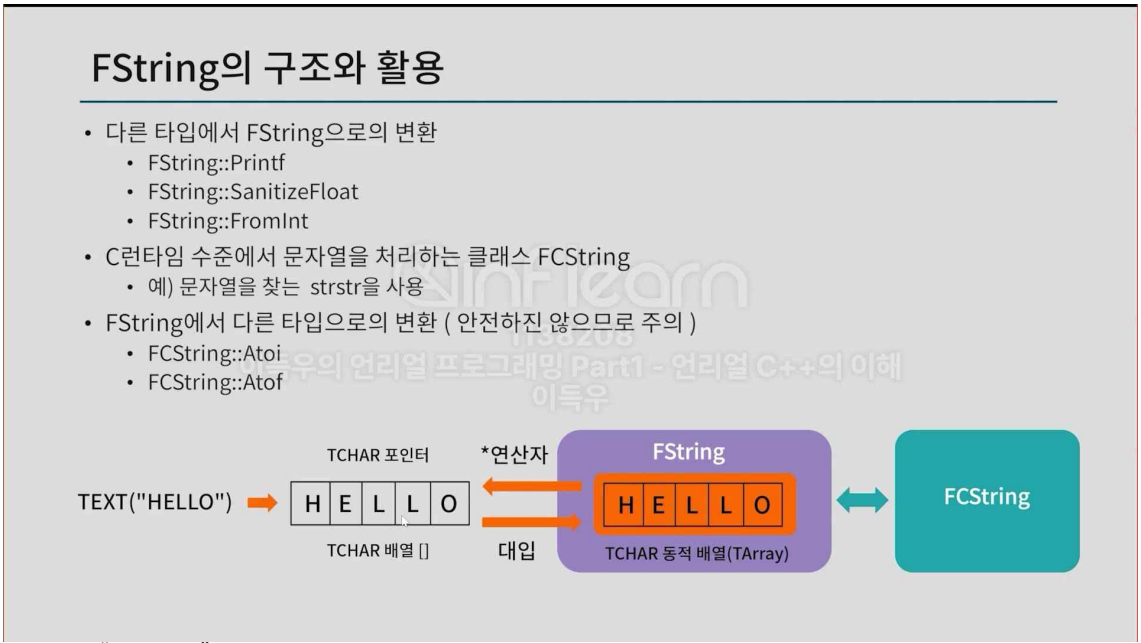
-이러한 문자열을 다루는 클래스로 FString을 사용하여 다양한 조작이 가능하다

**FString 공식 문서

-링크 : https://dev.epicgames.com/documentation/ko-kr/unreal-engine/fstring-in-unreal-engine?application_version=5.1

- FString TestHUDString만 선언하면 내부에 텅빈 껍데기가 생기고, FString TestHUDString = FString(TEXT("This is my test FString.")); --> 이런식으로 TCHAR의 동적 배열을 소유하고 있는 클래스임을 나타낸다.
- 대부분이 구조체들이 ToString이라는 함수를 제공해서 FString으로 변환되는 API를 제공한다. --> 디버깅할 때 각 데이터 값을 편리하게 볼 수 있다.
- 비교, 검색, 조립, printf등도 FString이 지원한다.
- 2바이트 문자열을 1바이트 문자열로 변환하는것도 제공한다.

**FString의 구조와 활용



-TEXT(“HELLO”) 와 같이 선언하면 TCHAR 배열로 만들어지게 된다.
-이것을 FString으로 집어넣는 순간 TArray라는 동적배열 클래스가 TArray방식으로 HELLO가 보간이 된다. 그러면 이 동적배열에서 데이터를 꺼내면 동적 배열이 속하고 있는 내부자료에 대한 포인터를 가져와서, 이걸 디 레퍼런싱이라고 한다. 즉 넘겨줄 때 사용하는 것이 포인터(*) 연산자이다.
-포인터 연산자를 써주게 된다면 FString에 포함하고 있는 동적배열 ,TArray가 포함하고 있는 첫 번째 인자의 포인터를 반환해준다. 이렇게 우리가 FString을 중심으로 TCHAR을 관리하되, TCHAR형태로 데이터가 필요한 경우에는 포인터 연산자를 써서 무리없이 사용이 가능하다.
-FString자체적으로 다양한 기능을 제공하기 때문에 안에 있는 데이터를 자르거나 붙이거나 새로운 방식으로 불르는 다양한 연산을 할 수 있다.
-FString이 제공하는 함수들은 내부적으로 FCString이라는 클래스가 있는데 일종의 C라이브

러에서 제공하는 기본적인 스트링 관련 함수들을 포함하고 있는 레퍼 클래스이다. 즉 실제 문자열을 찾거나 자르는 처리들은 FCString을 통해 진행을 한다.
-사진에서는 많이 쓰는 기능들을 각각 정리해둔 것이다.
-저수준 라이브러리 경우에는 포인터 연산을 하기 때문에 안전하다고 보장하지 않기 때문에 사용시 주의를 해야한다.

FString API 테스트 및 예제

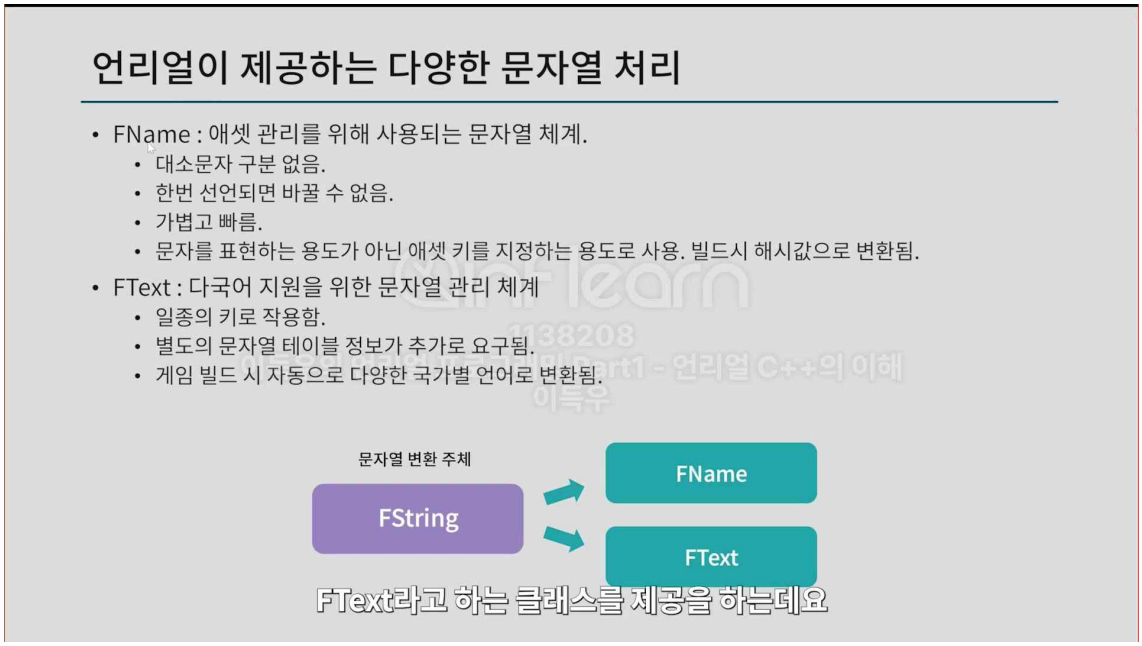
```
void UMyGameInstance::Init()
{
	Super::Init();
	TCHAR LogCharArray[] = TEXT("Hello Unreal");
	UE_LOG(LogTemp, Log, LogCharArray);

	FString LogCharString = LogCharArray;
	UE_LOG(LogTemp, Log, TEXT("%s"), LogCharArray);


	// ** 예시1 문자열 끄집어 내느 포인터 연산자 예시**//

	const TCHAR* LongCharPtr = *LogCharString; //포인터 연산자를 사용하면 TCHAR 포인터로 들어오게 되고, 이것은 불변하는걸로 const를 붙여 고칠 수 없는 형태로 참조하도록 선언이 되어 있다.
	TCHAR* LogCharDataPtr = LogCharString.GetCharArray().GetData(); //포인터를 받고싶은데 Const가 아닌 직접 고치고 싶을때
	//GetCharArray()==>캐릭터 어레이를 직접 가져올 수 있다.
	//GetData()==> 실제 TCHAR 포인터를 가져와서 const를 안붙었기 때문에 여러가지 메모리에 직접 접근하여 사용이 가능하다./

	TCHAR LogCharArrayWithSize[100]; //배열로 직접 가져오고 싶은 경우 저수준의 스트링 복사가 필요하다.
	FCString::Strcpy(LogCharArrayWithSize, LogCharString.Len(), *LogCharString);
	//저수준 API 대응 함수들은 대부분 FCString에서 사용되어지고 있다. , 복사를 해주되, 버퍼를 지정해줘야 한다.)

	if (LogCharString.Contains(TEXT("unreal"), ESearchCase::IgnoreCase))
		// 대소문자를 구분해서 비교할것인지 상관없이 비교할것인지 옵션지정 ==> ESearchCase, IgnoreCase==> 대소문자 구분 없이 진행
	{
		int32 Index = LogCharString.Find(TEXT("unreal"), ESearchCase::IgnoreCase);
		// 찾아 본다 unreal이라는 단어를 찾아주면 인덱스를 뽑아줄수있다.==> Find [몇번째 위치에 unreal이라는 문자가 있는지]
		FString EndString = LogCharString.Mid(Index);
		// 이부분이 자르는 곳인데 Index를 넣어주면 unreal이 시작되는 위치부터 끝까지 잘라주게 된다.
		UE_LOG(LogTemp, Log, TEXT("Find Text: %s"), *EndString);
		// 로그를 한번 찍어본다. (여기서 실수할 수 있는 부분이 항상 인자로 들어갈땐 포인터(*) 연산자 넣어주자
	}
```

```
	// ** 예시2 문자열 자르는 함수 예시 **//

	FString Left, Right;
	if (LogCharString.Split(TEXT(" "), &Left, &Right)) // 공백을 기점으로 왼쪽 오른쪽을 나눔
	{

		UE_LOG(LogTemp, Log, TEXT("Split Text: %s 와 %s"), *Left, *Right);
		//여기까지만 쓰고 실행하면 와==? 로 뜨는것을 볼 수 있음
		//?가 뜨는 이유는 윈도우에서 한글을 썻기 때문에 CP949형태에 멀티바이트 스트링으로 저장이되서 지금 UTF-16을쓰는 언리얼과 호환이 안되는거임
		// File->Save AS->SAVE옆 화살표-> Save with Encoding -> Unocode(UTF-8 with Signature) [여기서 Signature는 BOM정보가 들어있는 UTF-8형식을 의미함->OK
	}

	// ** 예시3 변환하는 함수 예시 예시 **//
	int32 IntValue = 32;
	float FloatValue = 3.141592;
	FString FloatIntString = FString::Printf(TEXT("Int:%d Float:%f"), IntValue, FloatValue);
	//Printf문을 사용해서 두가지를 한번에 출력하는게 가장 편하긴 함

	FString FloatString = FString::SanitizeFloat(FloatValue);//만약 내가 이 단일 Value를 String으로 바꾸고 싶다면
	FString IntString = FString::FromInt(IntValue); //Int형 단일 변환이 가능하다.

	UE_LOG(LogTemp, Log, TEXT("%s"), *FloatIntString);
	UE_LOG(LogTemp, Log, TEXT("Int:%s Float:%s"), *IntString ,*FloatString);


	//** 예시4 FloatString을 다시 Integer로 변환하기 **//
	int32 IntValueFromString = FCString::Atoi(*IntString);
	float FloatValuyeFromString = FCString::Atof(*FloatString);
	FString FloatIntString2 = FString::Printf(TEXT("Int:%d Float:%f"), IntValueFromString, FloatValue);
	UE_LOG(LogTemp, Log, TEXT("%s"), *FloatIntString2);
}
```

-기존고 다른 포인터 연산 사용에 어색할수도 있지만, 왜 포인터연산을 사용하는지, FString구조가 어떤 내부구조를 이루는지 이해하여 사용하도록 하자

FNAME 클래스 활용

**언리얼이 제공하는 다양한 문자열 처리



-FNAME같은 경우 에셋관리를 위해 사용되는 문자열 체계이다. [해쉬값(key-value쌍) 사용하며 구조를 따로 제공함]
-FNAME은 대소문자 구분이 없고 한번 선언하면 key라 int로 변환이 된다. 따라서FString처럼 문자열을 바꾸거나 조절이 안되며 바꿀려면 FString으로 변환해야 하는데 FNAME은 대소문자 구분이 없어 데이터가 깨질 수 있다.

-FText는 UI에서 다국어 지원을 할 때 필요한 문자 체계이다.

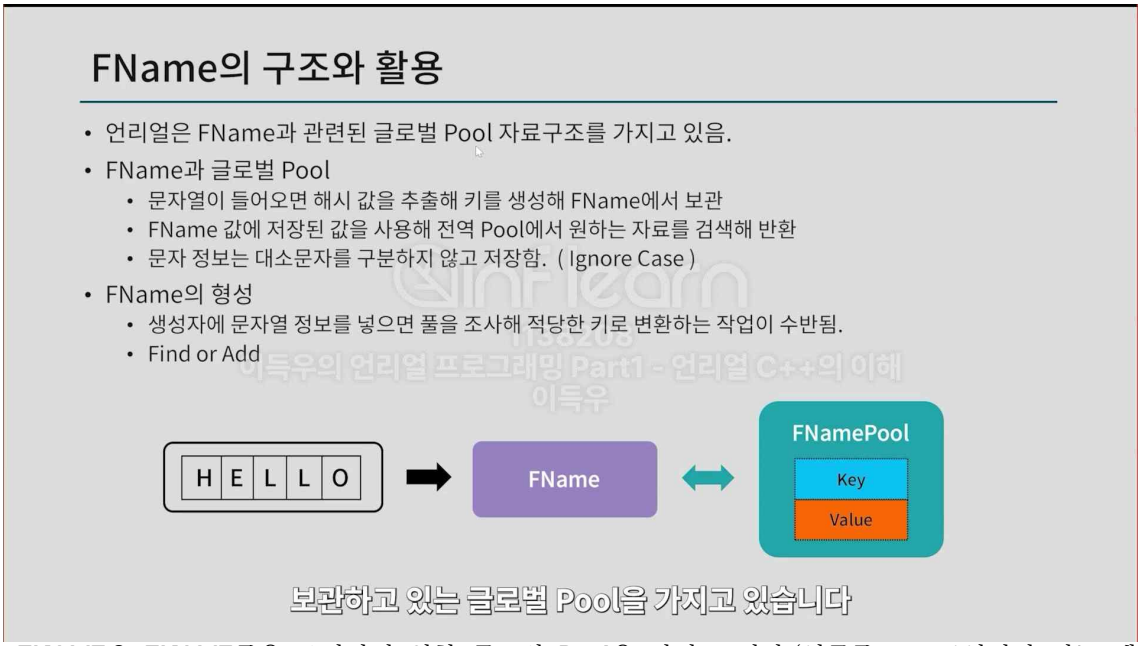
-정리: FString을 사용하여 문자를 관리하는데 FNAME이나 FText로 변환하여 다양한 용도로 활용이 가능하다.

**FNAME 공식 문서

-링크 : https://dev.epicgames.com/documentation/ko-kr/unreal-engine/fname-in-unreal-engine?application_version=5.1

-FNAME은 초경량 시스템으로써 해시테이블을 사용해서 빠르다. (찾기위한 용도이며 문자열을 처리하기 위한 용도가 아님.)
-내부적으로는 결국 인덱스값이 저장되어 있다(인덱스Key값이 FName의 실체)

**FNAME 구조와 활용



-FNAME은 FNAME들을 보관하기 위한 글로벌 Pool을 가지고 있다.(싱글톤으로 구성되어 있는 네임드만 모아둔 자료구조임)[key-value쌍으로 들어가 있음]

-우리가 문자열 정보를 넣었을 때 문자열 정보는 따로 저장이 되고, 문자열을 변환한 해쉬값 즉 key가 저장이 된다.

-우리가 key값을 사용하여 pool에 있는지 없는지 검출하고 있다면 그 값을 사용하여 우리가 원하는 데이터를 가져오는 형태로 구성이 되어 있다.

-경량으로 빠르게 처리 가능한 문자열 시스템이다.

-FNAME에는 실제 자료는 key값만 들어가 있기 때문에 용량이 작다.

-용도로는 pool 데이터가 있는지 찾거나 없으면 추가하는 등의 행동만 가능하다.

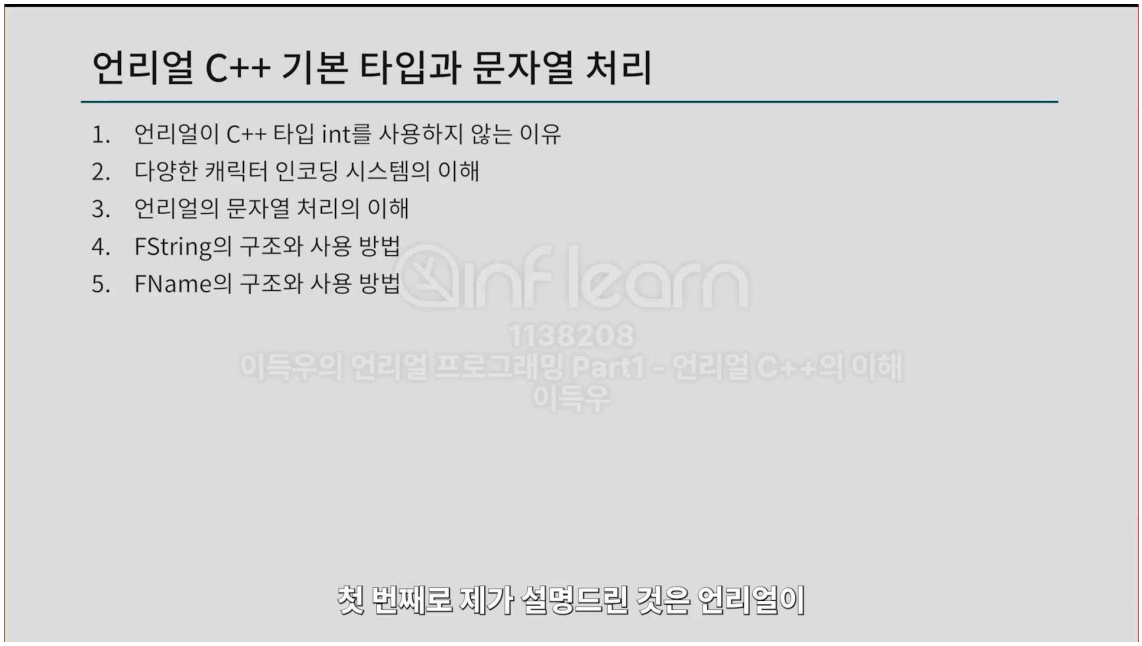
-우리가 에셋들을 보관하는데 유용하게 활용이 가능하다

-실제 게임을 제작할 때 이런 FName을 사용하여 이런 key값을 사용하는 문자열들을 관리하는 것이 좋다.

**FNAME 예제

```
void UMyGameInstance::Init()
{
	//**예제1: FName인 대소문자 구분이 없음**//
	FName key1(TEXT("PELVIS"));
	FName key2(TEXT("pelvis"));
	UE_LOG(LogTemp, Log, TEXT("FNAME 비교결과 : %s"), key1 == key2 ? TEXT("같음") : TEXT("다름"));


	//**예제2: FName 구성 방법
	for (int i = 0; i < 10000; ++i)
	{
		FName SearchInNamePool = FName(TEXT("pelvis"));
		/*
		이렇게 선언하게 되면, 생성자에 이
		문자열을 넣으면 FNAME은 문자열을 key로 변환하여
		key가 전역풀에 있는지 조사하는 작업을 거치게 되며
		이렇게 빈번한 작업이 일어나는데 있어서 오버헤드가 발생할 수 있다.
		결과적으로 조사해서 FName에 관련 key값만 저장하면 됨
		전에 한번만 선언해주거나 const이용을 하면 된다.
		*/

		const static FName StaticOnlyOnce(TEXT("pelvis"));
		/*
		이렇게하면 처음 초기화할때 데이터를 저장하고,
		local static으로 선언했으니 그 다음부터 찾을일이 없다.

		나중에 게임 만들대 FName이 Tick과 같이 자주 실행되는 함수에 있는경우
		오버해드가 발생할 수 있으니 잘 알아두자.
		*/
	}
}
```

**이번 강의 정리


언리얼 C++ 기본 타입과 문자열 처리
1. 언리얼이 C++ 타입 int를 사용하지 않는 이유
2. 다양한 캐릭터 인코딩 시스템의 이해
3. 언리얼의 문자열 처리의 이해
4. FString의 구조와 사용 방법
5. FName의 구조와 사용 방법
inflearn
1138208
이득우의 언리얼 프로그래밍 Part1 - 언리얼 C++의 이해
이득우
첫 번째로 제가 설명드린 것은 언리얼이