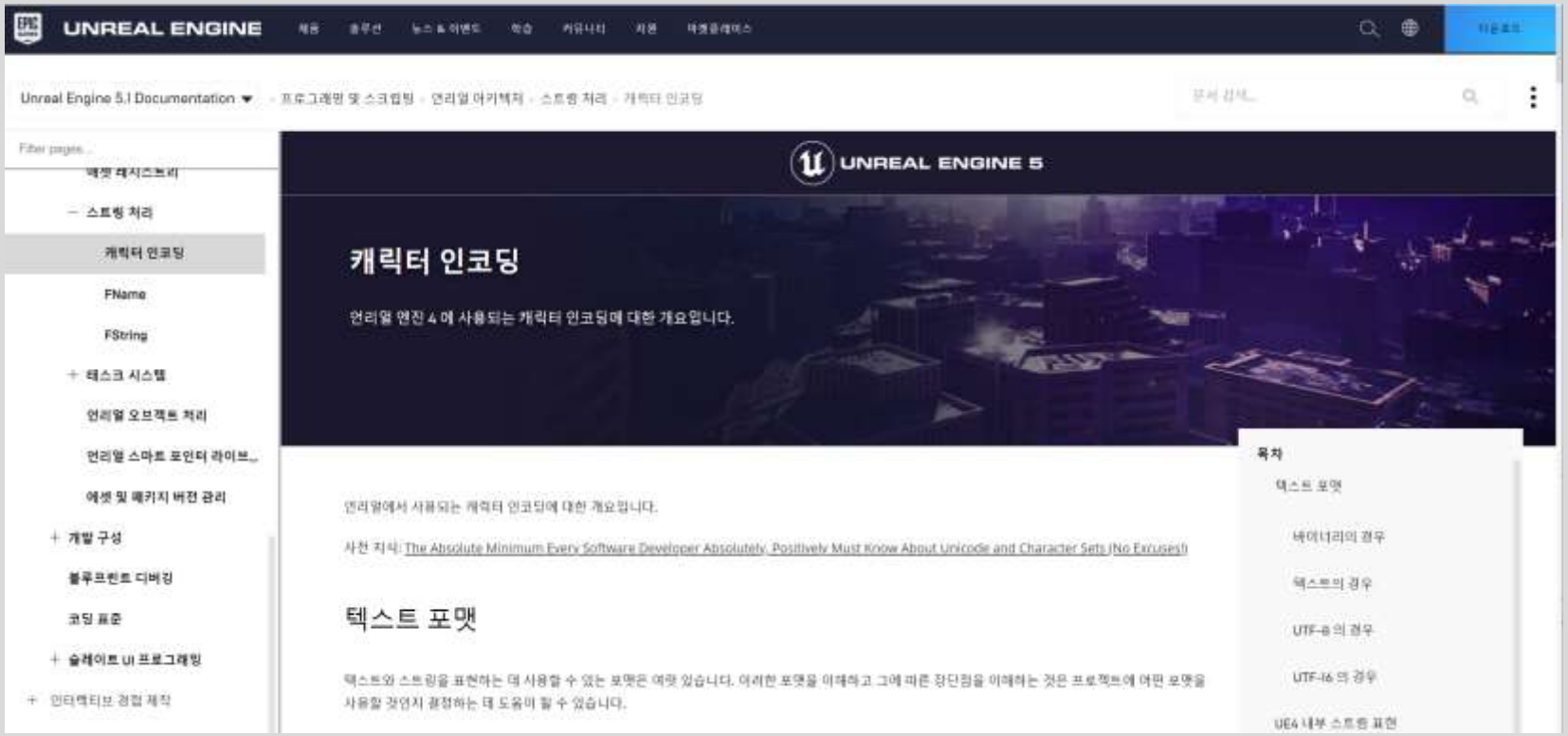
언리얼 C++ 기본 타입과 문자열

(Unreal C++ primitive types and string)

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++ 기본 타입

왜 언리얼은 기본 타입을 따로 지정하는가?

- 1970년대에 개발되서 아직도 사용 중인 C++ 언어
  - 시대에 따라 발전한 하드웨어 사양
  - 플랫폼 파편화(Platform Fragmentation)
- C++ 최신 규약에서 int는 최소 32비트를 보장하도록 규정되어 있음.
  - 특정 플랫폼에서는 64bit로 해석될 수 있음.
  - 따라서 데이터를 저장할 때 int 타입의 크기를 확신할 수 없음
- 게임 제작의 특징
  - 데이터 정보가 명확해야 한다.
  - 단일 컴퓨터에서 최대 퍼포먼스를 뽑아내야 한다.
  - 네트웍 상에서 데이터 통신이 효율적이고 안정적이어야 한다.

데이터 타입의 애매 모호함은 게임 개발시 문제를 일으킬 수 있음.

int 타입과 크기

- 후발 언어 C#의 경우 int 타입이 있지만 4바이트인 int32로 명확히 정의되어 있음
- 언리얼은 int를 사용하지 않고 int32를 사용함

Characteristics of the integral types

C# supports the following predefined integral types:

C# type/keyword	Range	Size	.NET type
`sbyte`	-128 to 127	Signed 8-bit integer	System.SByte
`byte`	0 to 255	Unsigned 8-bit integer	System.Byte
`short`	-32,768 to 32,767	Signed 16-bit integer	System.Int16
`ushort`	0 to 65,535	Unsigned 16-bit integer	System.UInt16
`int`	-2,147,483,648 to 2,147,483,647	Signed 32-bit integer	System.Int32
`uint`	0 to 4,294,967,295	Unsigned 32-bit integer	System.UInt32
`long`	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer	System.Int64
`ulong`	0 to 18,446,744,073,709,551,615	Unsigned 64-bit integer	System.UInt64
`nint`	Depends on platform (computed at runtime)	Signed 32-bit or 64-bit integer	System.IntPtr
`nuint`	Depends on platform (computed at runtime)	Unsigned 32-bit or 64-bit integer	System.UIntPtr

→ **int8 or uint8**

→ **int16 or uint16**

→ **int32 or uint32**

→ **int64 or uint64**

→ 포인터 크기는 불분명(추정하지 말 것!)

출처 : https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/integral-numeric-types

언리얼 엔진의 타입과 크기

- 언리얼 엔진에서 사용하는 기본 타입 (코딩 표준 문서 참고)

포팅 가능한 C++ 코드

- `bool` - boolean 값(bool 크기 추정 금지). `BOOL` 은 컴파일되지 않습니다.
- `TCHAR` - character(TCHAR 크기 추정 금지)
- `uint8` - unsigned byte(1 바이트)
- `int8` - signed byte (1 바이트)
- `uint16` - unsigned "short"(2 바이트)
- `int16` - signed "short"(2 바이트)
- `uint32` - unsigned int(4 바이트)
- `int32` - signed int(4 바이트)
- `uint64` - unsigned "quad word"(8 바이트)
- `int64` - signed "quad word"(8 바이트)
- `float` - single precision floating point(4 바이트)
- `double` - double precision floating point(8 바이트)
- PTRINT - 포인터를 가질 수 있는 integer(PTRINT 크기 추정 금지)

C++ 의 `int` 와 unsigned `int` 유형은 플랫폼에 따라 크기가 변할 수 있지만 너비가 적어도 32비트임이 보장되므로 정수 너비가 중요치 않은 경우라면 코드에서 사용해도 괜찮습니다. 명시적으로 크기가 정해진 유형은 여전히 시리얼라이즈 또는 리플리케이트된 포맷으로 사용해야 합니다.

bool 타입의 선언

- 데이터 전송을 고려한 참/거짓 데이터의 지정
- bool은 크기가 명확하지 않음.
- 헤더에는 가급적 bool 대신 uint8 타입을 사용하되 Bit Field 오퍼레이터를 사용.
- 일반 uint8과의 구분을 위해 b접두사를 사용.
- Cpp 로직에서는 자유롭게 bool을 사용

```
/** If true, when the actor is spawned it will be sent to the client but receive no further
UPROPERTY()
uint8 bNetTemporary:1;

/** If true, this actor was loaded directly from the map, and for networking purposes can be
uint8 bNetStartup:1;

/** If true, this actor is only relevant to its owner. If this flag is changed during play,
UPROPERTY(Category=Replication, EditDefaultsOnly, BlueprintReadOnly)
uint8 bOnlyRelevantToOwner:1;

/** Always relevant for network (overrides bOnlyRelevantToOwner). */
UPROPERTY(Category=Replication, EditDefaultsOnly, BlueprintReadWrite)
uint8 bAlwaysRelevant:1;
```

캐릭터 인코딩

왜 언리얼은 문자열을 따로 지정하는가?

- 1990년대 후반에 이르러서야 표준화된 아시안 문자열 표준 (한국, 중국, 일본)
- 하지만 컴퓨터는 그전에도 사용했었다.
- 문자열 처리의 종류
  - Single byte(ANSI, ASCII) : 컴퓨터 초창기
  - Multibyte(EUC-KR, CP949) : 컴퓨터 보급기 (1990년대 초중반)
  - Unicode(UTF-8, UTF-16) : 국제 표준 정착기 (1990년대 후반)
- 하지만 이 모든 문자열은 아직도 사용되고 있음
  - C++ STL은 ASCII, UTF-8, UTF-16만 지원함.
  - Windows 10은 멀티바이트를 지원함. 하지만 다른 운영체제는 지원하지 않음.

캐릭터 인코딩

https://bit.ly/uecharen

https://docs.unrealengine.com/5.1/ko/character-encoding-in-unreal-engine/

TCHAR와 FString

복잡한 문자열 처리를 하나로.

- 유니코드를 사용해 문자열 처리를 통일.
  - 이 중에서 2byte로 사이즈가 균일한 UTF-16을 사용
  - 유니코드를 위한 언리얼 표준 캐릭터 타입 : TCHAR
- 문자열은 언제나 TEXT 매크로를 사용해 지정.
  - TEXT매크로로 감싼 문자열은 TCHAR 배열로 지정됨.
- 문자열을 다루는 클래스로 FString를 제공함
  - FString은 TCHAR 배열을 포함하는 헬퍼 클래스

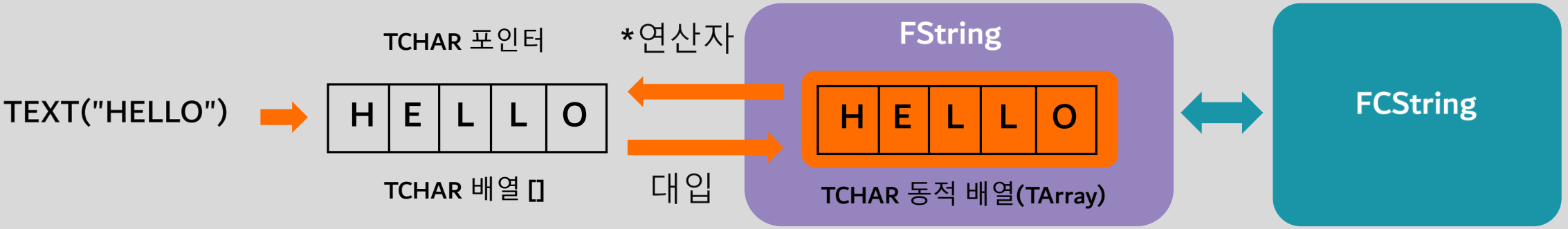
FString 클래스

https://bit.ly/uefstringkr

https://docs.unrealengine.com/5.1/ko/fstring-in-unreal-engine/

FString의 구조와 활용

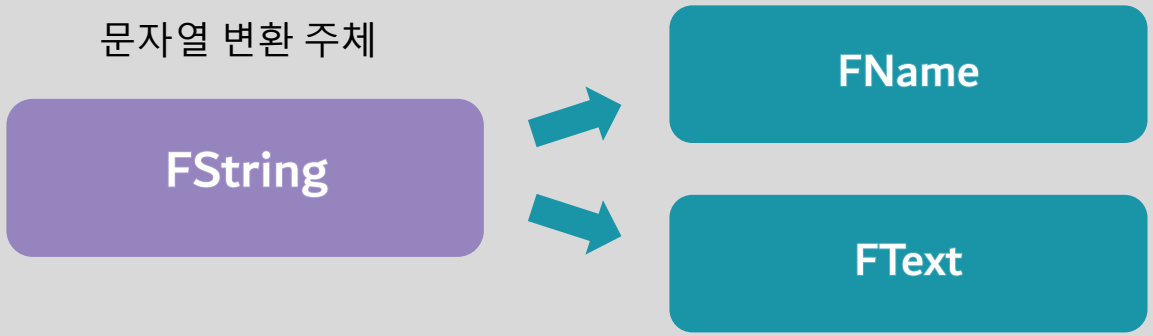
- 다른 타입에서 FString으로의 변환
  - FString::Printf
  - FString::SanitizeFloat
  - FString::FromInt
- C런타임 수준에서 문자열을 처리하는 클래스 FCString
  - 예) 문자열을 찾는 strstr을 사용
- FString에서 다른 타입으로의 변환 (안전하진 않으므로 주의)
  - FCString::Atoi
  - FCString::Atof



FName의 활용

언리얼이 제공하는 다양한 문자열 처리

- FName : 애셋 관리를 위해 사용되는 문자열 체계.
  - 대소문자 구분 없음.
  - 한번 선언되면 바꿀 수 없음.
  - 가볍고 빠름.
  - 문자를 표현하는 용도가 아닌 애셋 키를 지정하는 용도로 사용. 빌드시 해시값으로 변환됨.
- FText : 다국어 지원을 위한 문자열 관리 체계
  - 일종의 키로 작용함.
  - 별도의 문자열 테이블 정보가 추가로 요구됨.
  - 게임 빌드 시 자동으로 다양한 국가별 언어로 변환됨.



FName 클래스

https://bit.ly/uefnamekr

https://docs.unrealengine.com/5.1/ko/fname-in-unreal-engine/

FName의 구조와 활용

- 언리얼은 FName과 관련된 글로벌 Pool 자료구조를 가지고 있음.
- FName과 글로벌 Pool
  - 문자열이 들어오면 해시 값을 추출해 키를 생성해 FName에서 보관
  - FName 값에 저장된 값을 사용해 전역 Pool에서 원하는 자료를 검색해 반환
  - 문자 정보는 대소문자를 구분하지 않고 저장함. (Ignore Case)
- FName의 형성
  - 생성자에 문자열 정보를 넣으면 풀을 조사해 적당한 키로 변환하는 작업이 수반됨.
  - Find or Add

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