Event Driven C는 이밴트 기반 C언어이다. 게임과 같은 가상현실 세계 개발에 목적을 가진 언어이다. C++은 객체 지향언어이지만, 각 객체만의 흐름을 가지고 있지 않다.  
현실세계는 절차적으로 작동하지 않기 때문에, 현실세계와 같은 가상 공간을 구현하기 위해서는  
Multi Threading을 하거나, Coroutine 같은 것들로 concurrency를 구현해야 한다.  
C++에서는 이러한 동시성을 구현하기 위해 쓰래드풀과 data race condition을 막기 위한 어려가지 작업을 필요로 하기 때문에, 언어차원에서 게임엔진의 업데이트 함수 같은 것과 병렬 프로그래밍에 필요한 요소들을 제공하여 concurrency 프로그래밍 작업과, interactive app을 개발하기 쉽게 하기 위한 목적을 가지고 있다. C/C++은 강력하지만 실수하기 쉬운, 위험한 언어이다. 그래서, C/C++의 장점을 최대한 수용하고, 개발자가 언어를 사용하면서 실수하기 쉬운 위험한 기능을 개선 및 제거하는 방식으로 언어를 디자인 할 생각이다.  
기본적인 syntex는 C++과 매우 비슷하며, 몇가지 사항만 다르다.

EDC는 동시성을 지원하는 객체지향 언어이다.

1. 컴파일 언어이다.  
2. nullptr integrity를 보장한다. 런타임에 잘못된 주소 참조를 통한 에러가 없다는 뜻이다.  
null 참조 불가.  
3. 범위 기반 가비지 컬랙션을 지원한다.  
4. 모든 변수는 원자성이 보장된다.  
5. 모든 객체는 mutex가 내장되어 있다.

상수 데이터 타입 종류:

문자형:   
**char** [1byte], **string** [dynamic]  
**wchar** [4byte], **wstring** [dynamic]

정수형:   
**sbyte**[1byte / int8], **ubyte**[1byte / uint8]  
**short**[2byte / int16], **ushort**[2byte / uint16]  
**int**[4byte / int32], **uint**[4byte / uint32]  
**long**[8byte / int64], **ulong**[8byte / uint64]  
**~~int128~~** ~~[16byte],~~ **~~uint128~~** ~~[16byte]~~

논리형:  
**boolean**[1byte]

/\* 부동소수점 정확도가 높은 대신 메모리 많이 사용함 \*/  
부동소수점형:  
**float32** [8byte], **ufloat32** [8byte]  
**float64** [16byte], **ufloat64** [16byte]

변수형:  
var int a; //versatile 키워드 사용시 변수로 사용 가능. 없으면 상수

특수형:  
**generic** [16byte] 모든 데이터 타입 담을 수 있음.

**unique generic** rainbow; // ok.

포인터:  
**function** [8byte] 모든 타입의 함수 포인터임  
**method** [8byte] 객체의 맴버 함수 포인터

\*로 역참조하여 값만 복사가능.  
& 사용시 포인터의 주소값 반환  
&\* 사용시 역참조한 공간의 주소를 반환

**unique** [exclusive ownership pointer / ...byte]  
example code:   
**int** main()  
{  
 // heap에 동적할당  
 {  
 **unique int** x = var 5; // 변수 int를 가르키는 상수 unique pointer  
 **var unique int** y = 7; // 상수 7을 가르키는 unique pointer 변수  
 } // nullptr이 아니기 때문에 동적 할당 해제, 메모리 반환.  
 return 0;   
}

**int** main()  
{  
 **var unique int** z = var 0; // 초기화 시, 동적할당. 만약 unique int z; 이라면, 동적할당이 일어나지 않음  
 // heap에 동적할당  
 {  
 **unique int** x = 5;  
 z = x; // 소유권 넘김. z에 있는 공간은 예약상태로 전환 후, 다른 unique 할당시 초기화 하여 사용.  
 } // x는 소유권을 넘겨서 nullptr이기 때문에 heap 반환 없음.

print(z); // 5출력됨.  
 return 0;   
}

**indirect** [ indirection pointer / ...byte] // has invalid address sanitizer  
example code:   
**int** main()  
{  
 **var indirect int** lurker; // 간접참조만 가능.  
 {  
 **unique int** x = 5; //상수 포인터에 heap 스택머신으로 상수 값이 있는 공간 주소 할당  
 lurker = x; // lurk x  
 } // nullptr이 아니기 때문에 x 동적 할당 해제, 메모리 반환.  
 return 0;   
}

**int** main()  
{  
 **indirect int** lurker = 0; // ERROR! CAN NOT ALLOCATE MEMORY!  
 return 0;   
}

int main()  
{  
 **indirect generic** a; // OK. indirects generic object.  
 **indirect unique** b; // OK. indirects unique pointer [not the object that’s pointed by unique]  
 **indirect unique generic** c; // OK. indirects unique generic pointer.  
 **int** d[]; // Static Stack Array  
 **unique int** e[]; // Dynamic Heap Array.  
 **unique int** f[][]; // Dynamic 2D Heap Array.   
 **unique int** g[][][]; // Dynamic 3D Heap Array.   
 **unique indirect** h; // OK. unique pointer for indirection pointer

**generic indirect** i; // ERROR! ILLEGAL EXPRESSION

/\* bad types example: function indirect \*/  
 return 0;   
}

**cache** [suggest allocating a variable on CPU cache memory] // does not support arrays  
example code:  
int main()  
{  
 **indirect generic cache** a; // ERROR.  
 **cache indirect unique** b; // OK.  
 **indirect cache unique generic** c; // OK.   
 **cache int** d[]; // Static Stack Array  
 **unique cache int** e[]; // ERROR.  
 **cache unique int** f[][]; // ERROR.  
 **unique cache int** g[][][]; // ERROR.  
 **unique cache indirect** h; // OK.   
 **cache generic** i; // OK.  
 return 0;   
}

추가 기본 제공함수:  
**void** pre\_tick();  
**void** tick();  
**void** post\_tick();  
**void** post\_constructor();  
**void** pre\_destructor();

// Event Driven C main() function

#include "std.edc"

namespace system\_main

{

static int main()

{

return 0;

}

}

class: 맴버변수와 함수를 가질 수 있다.  
struct: 맴버변수만 소유할 수 있다.  
... 나머지는 C++과 동일하다.