OOP With C#

C# 형식,기본문법

1.기본 자료형 (primitive type)

정수, 부울

타입명	크기	표현범위
sbyte	8 bit	-128 ~ 127
byte	8 bit	0 ~ 255
short	16 bit	-32768 ~ 32767
ushort	16 bit	0 ~ 65536
int	32 bit	-2147483648 ~ 2147483647
uint	32 bit	0 ~ 4294967295
long	64 bit	-9223372036854775808 ~ 9223372036854775807
ulong	64 bit	0 ~ 18446744073709551615
bool	8 bit	true , false

문자 (character)

타입명	크기	표현범위
char	16 bit	U+0000 ~ U+FFFF

문자 타입은 각 문자 (알파벳 등) 를 숫자에 대응하는 표준에 따라 그 값을 저장하며,
이 때 사용되는 표준을 유니코드 라 한다.

U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0010	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0020	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0050	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0060	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0070	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

유니코드에서 로마자 기본 영역이며, ASCII코드의 영역이다

대표적인 문자코드

숫자0~9 U+0030 ~ U+0039
알파벳(대) U+0041 ~ U+005A
알파벳(소) U+0061 ~ U+007A
한글자음 U+1100 ~ U+1112
한글모음 U+1161 ~ U+1175
한글완성 U+AC00 ~ U+D7A3 (가 ~ 힣)

문자 타입은 문자코드표에 따라 정수 타입과 바로 변환이 가능하다

입력 >

```
char aFromChar  = 'a';
char aFromInt   = (char)0x0061;
int  intFromA   = (int)'a';
```

출력 >

```
aFromChar   : a
aFromInt    : a
intFromA    : 97 ( hex 0x0061 )
```

부동소수점

타입명	크기	표현범위
float	32 bit	$\pm 1.5 \times 10^{-45} \sim \pm 3.4 \times 10^{38}$
double	64 bit	$\pm 5.0 \times 10^{-324} \sim \pm 1.7 \times 10^{308}$

1) 구조 (32 bit)

부호비트 1 , 지수비트 8 , 가수비트 23

2) IEEE 754 부동소수점 변환법

1. 정수부, 소수부를 각각2진수로 변환
2. 2진수로 변환된 정수부 소수부를 연달아 표현
3. 해당 숫자의 맨 앞 1이 소수점 바로 앞까지 오는데 필요한 2의 배수를 계산
4. 3에서 계산된 2의 배수를 2의 승수로 표현하고, 이 지수에 127을 더해 지수비트에 기입
5. 과정 3에서 남은 소수점 아래 모든 숫자를 가수비트에 기입
6. 음수의 경우 부호비트를 1로 기입

실수를 부동소수점으로 표현 하는 예) 3.14

3 -> 11

0.14 -> 00100011111010111000 …

11.00100011111010111000 -> 2^1

1.10010001111010111000

부호 비트 : 양수이므로 0

지수 비트 : 01111111 + 1 = 10000000

가수 비트 : 1001 0001 1110 1011 1000 000

* 소수점 2진수 변환방법

숫자에 2를 곱하여 나온 정수부를 연달아 기입한다 (다음 연산에서 정수부는 버린다)

0.14 x 2 = 0.28

0.28 x 2 = 0.56

0.56 x 2 = 1.12

0.12 x 2 = 0.24 => 00100011 …

0.24 x 2 = 0.48

0.48 x 2 = 0.96

0.96 x 2 = 1.92

0.92 x 2 = 1.84 …

열거형

- 정수 상수에 이름을 부여하여 사용하는 타입
- 가독성을 위해 사용한다

일반 선언

```
enum eServerType
{
	TEST,
	DEV,
	LIVE
}
```

상수를 명시

```
enum eServerType
{
	TEST	= 10,
	DEV	= 55,
	LIVE	= 100
}
```

- 상수를 명시하는 경우 사용 예

  게임서버 주소 = “101.102.103.”+(int)eServerType.TEST

2.연산자

산술연산자

사칙연산	+ , - , * , /
모듈러	%
단항 증감 연산	++ , —
단항 연산	+ , -
복합 할당	+= , -= , *= , /=

연산자 우선순위

1. 후위 증감 연산
2. 전위 증감 연산, 단항 연산
3. 곱셈 및 나눗셈 , 모듈러연산
4. 덧셈 및 뺄셈 연산

- 우선순위가 높은것부터 계산하며, 같은 우선순위를 가진 연산에서는 왼쪽부터 계산한다
- 괄호 () 가 있는 경우 괄호 안의 연산이 우선순위를 가진다

논리연산자

& , ^ , | , || , &&

- bool 형에 대해 AND , OR , XOR 연산
- 정수형에 대해 이진 AND, OR ,XOR 연산

!

- bool 형에 대해 논리 부정 연산

비트 및 시프트 연산자

~

- 정수형에 대해 보수 (1의 보수) 를 생성

<< , >>

- 오른쪽 피연산자 만큼(횟수) 왼쪽 피연산자의 비트를 이동

* 2진수 결과 출력을 위해서 아래와 같은 보간 문자열을 사용하시면 됩니다
 $”{Convert.ToString(var, toBase:2)}”

비교 연산자

> , < , >= , <=

- 정수 또는 부동소수점 형식에 대해 크기 비교
- 문자 형식에 대해 문자 코드의 정수 크기 비교

같음(Equality) 연산자

== , !=

- 값 형식에 대해 두 변수의 값이 같거나 같지 않음을 평가
- 참조 형식에 대해 두 변수의 참조가 같거나 같지 않은지 평가

* 값 형식 : 변수에 데이터가 직접 포함
* 참조 형식 : 변수에 데이터의 참조가 포함

열거형과 비트 플래그

```
[Flags]
enum Edible
{
	Orange	= 0b_0000_0000,
	Beef	= 0b_0000_0001,
	Apple	= 0b_0000_0010,
	Fork	= 0b_0000_0100,
	Shrimp	= 0b_0000_1000
}
```

비트 플래그의 조합 예

```
Edible meat = Edible.Beef | Edible.Fork;
```

조합 된 비트 플래그의 연산 예

```
bool isShrimpMeat = (( meat & Edible.Shrimp ) == Edible.Shrimp )
```

3.구문

선택문

if 문	부울 표현식이 참인 경우 본문을 수행
if-else 문	부울 표현식에 의해 실행할 본문을 선택

```
if(aCondition)
{
    //code to execute when aCondition is true
}

if(aCondition)
{
    //code to execute when aCondition is true
}
else if(anotherCondition)
{
    //code to execute when anotherCondition is true
}
else
{
    //code to execute when neither aCondition nor anotherCondition are true
}
```

switch 문 입력된 인수를 각각의 식과 비교하여 일치된 본문 수행

```
switch(aValue)
{
    case > 10:
    case < -10:
        // code to execute when aValue is out of a given range of -10 to 10
        break;
    default:
        // code to execute when aValue is in a given range of -10 to 10
        break;
}
```

- break 키워드를 통해 switch 구문 블록을 탈출
- 중첩되지 않는 조건을 연달아 선언함으로써 부울 연산과 같은 효과를 낼 수 있다
- 패턴에 일치되지 않는 내용은 default 구문에서 처리 한다

반복문

for 문

- initializer , condition, iterator 와 본문으로 구성
  - Initializer : 루프가 실행 되기 전 한번만 실행되는 블록
  - Condition : 다음 루프를 실행할 지 판단하는 블록
  - Iterator : 루프가 실행 될 때 마다 수행될 내용 (카운터)

```
for( int i = 0 ; i < 10 ; i++ )
{
    // code to excute
}
```

- for 문의 initializer, condition, iterator 작성은 선택사항이다

```
// infinite loop
for(;;)
{

}
```

while 문

- 지정된 조건식이 true 인 동안 루프의 본문 실행
- 조건식이 true 인 동안, break 키워드를 통해 루프 탈출

```
bool runLoop = true;
while(runLoop)
{
    // code to excute
}
```

do-while 문

- 최초 한번은 본문 수행 후, 조건식 검사

```
bool runLoop = true;
do
{
    // code to excute
}while(runLoop);
```