C#
6.0

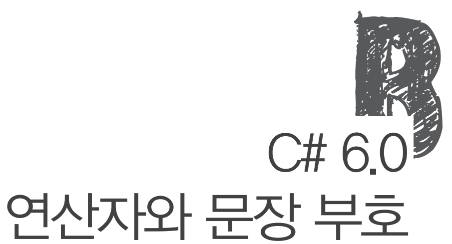
04

부록

A

C# 6.0 Language Specification Version 6.0 문서

비주얼 스튜디오 2015가 설치된 후 "C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC#\Specifications\1033" 폴더를 보면 "CSharp Language Specification.docx" 파일이 제공되는 데 이는 C# 5.0 문법만을 담고 있다. 마이크로소프트는 더 이상 이 문서를 업데이트하지 않기로 결정했고 대신 온라인에 정식 문서를 공개한다고 밝혔다(하지만, 아직 온라인 정식 문서는 없다).



C# 6.0 연산자와 문장 부호

부록 표 B.1: 연산자와 문장 부호

{	}	[	]	(	)	.	,	:	;
+	–	*	/	%	&	|	^	!	~
=	<	>	?	?. ?[] (C# 6.0)	??	::	++	––	&&
||	–>	==	!=	<=	>=	+=	–=	*=	/=
%=	&=	|=	^=	<<	<<=	=>			

부록 표 B.2: 연산자 우선순위

우선순위	연산자 범주	연산자
1	기본	x . y x?.y (C# 6.0 추가) f(x) a[x] a?[x] (C# 6.0 추가) x++ x–– new typeof checked unchecked default(T) delegate sizeof –>

<table>
<tr><th>우선순위</th><th>연산자 범주</th><th>연산자</th></tr>
<tr><td>2</td><td>단항</td><td>+x
−x
!x
~x
++x
−−x
(T)x
await
&x
*x</td></tr>
<tr><td>3</td><td>승제</td><td>x * y
x / y
x % y</td></tr>
<tr><td>4</td><td>가감</td><td>x + y
x − y</td></tr>
<tr><td>5</td><td>시프트</td><td>x << y
x >> y</td></tr>
<tr><td>6</td><td>관계 및 형식 테스트</td><td>x < y
x > y
x <= y
x >= y
is
as</td></tr>
<tr><td>7</td><td>비교</td><td>x == y
x != y</td></tr>
<tr><td>8</td><td>비트 논리곱</td><td>x & y</td></tr>
<tr><td>9</td><td>논리 XOR</td><td>x ^ y</td></tr>
<tr><td>10</td><td>비트 논리합</td><td>x | y</td></tr>
<tr><td>11</td><td>조건 논리곱</td><td>x && y</td></tr>
<tr><td>12</td><td>조건 논리합</td><td>x || y</td></tr>
<tr><td>13</td><td>Null 결합</td><td>x ?? y</td></tr>
<tr><td>14</td><td>3항 조건</td><td>?:</td></tr>
</table>

우선순위	연산자 범주	연산자
15	할당 및 람다 식	x = y x += y x −= y x *= y x /= y x %= y x &= y x |= y x ^= y x 〈〈= y x 〉〉= y =〉

* 1번부터 가장 우선순위가 높으며, 같은 그룹 내의 연산자는 우선순위가 같다.

C# 6.0 예약어

C# 6.0의 최신 예약어는 아래의 웹 사이트에서 확인할 수 있다.

```
C# Keywords
- http://msdn.microsoft.com/en-us/library/x53a06bb(v=vs.110).aspx
```

다음은 위 웹 사이트의 내용을 그대로 가져온 것이다.

부록 표 C.1: C# 6.0 키워드 79개

abstract	as	base	bool	break	byte
case	catch	char	checked	class	const
continue	decimal	default	delegate	do	double
else	enum	event	explicit	extern	false
finally	fixed	float	for	foreach	goto
if	implicit	in	in (generic modifier)	int	interface
internal	is	lock	long	namespace	new
null	object	operator	out	out (generic modifier)	override
params	private	protected	public	readonly	ref

return	sbyte	sealed	short	sizeof	stackalloc
static	string	struct	switch	this	throw
true	try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void	volatile
while					

부록 표 C.2: C# 6.0 문맥 키워드(Contextual Keywords) 25개

add	alias	ascending	async	await	descending
dynamic	from	get	global	group	into
join	let	orderby	partial (type)	partial (method)	remove
select	set	value	var	where (generic type constraint)	where (query clause)
yield					

ASCII 코드

10진수	ASCII	10진수	ASCII	10진수	ASCII	10진수	ASCII
0	NULL	32	SP	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	“	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	‘	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	–	77	M	109	m

10진수	ASCII	10진수	ASCII	10진수	ASCII	10진수	ASCII
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	₩	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

프로그래밍 기본 지식

프로그래밍 언어를 배우는 이유는 프로그램을 만들기 위해서다. 하지만 단순히 언어의 문법만을 배워서는 활용 사례가 극히 제한된다. 이를테면, 언어 자체는 다른 프로그램과의 통신을 지원하지 않기에 통신을 하려면 운영체제에서 제공하는 통신 수단을 알아야 할 필요가 있으므로 자연스럽게 운영체제와 관련된 지식 또한 필요하다. 따라서 여기서는 운영체제를 비롯해 프로그래밍할 때 익숙해져야 할 기본적인 용어를 알아본다.

E.1 하드웨어 관련 용어

E.1.1 중앙 처리 장치(CPU)

CPU(Central Processing Unit)는 고유한 기계어를 가지며, 그것을 해석할 수 있는 장치다. 일반적으로 윈도우 운영체제가 설치된 개인용 컴퓨터에는 대부분 인텔과 AMD에서 만든 CPU가 장착된다. 반면 최근 급증하는 모바일 기기에는 ARM이라는 회사에서 만든 CPU와 호환되는 제품이 주로 장착되고 있다. C#으로 만드는 프로그램들은 기본적으로 인텔 CPU와 호환되는 컴퓨터에서 실행되지만 ARM 계열에서 실행하는 것도 가능하다.

E.1.2 레지스터(Register)

CPU 내에 존재하는 기억장소다. 한 개의 레지스터에 담을 수 있는 데이터의 크기는 우리가 일반적으로 부르는 16비트/32비트/64비트 CPU라는 명칭에서 쉽게 알 수 있다. 즉, 64비트 CPU는 레지스터 하나에 담을 수 있는 비트의 수가 64개임을 의미한다. 인텔/AMD CPU에서는 하위 호환성을 확보하기 위해 16비트 응용 프로그램은 32비트/64비트 CPU에서도 실행할 수 있고, 32비트 역시 64비트 CPU에서 실행 가능하다.

CPU가 다르면 레지스터의 종류, 수, 이름도 바뀐다. 특히, 인텔 호환 CPU와 ARM 계열의 CPU는 완전히 상이한 레지스터를 가지고 있다. 지원되는 명령어 집합과 레지스터의 차이로 인해 인텔 호환 CPU에서 실행되는 응용 프로그램은 ARM 계열의 CPU에서 실행되지 않는다.

여러분이 만들게 될 C# 프로그램은 디스크 상에 파일로 존재하다가 실행되는 시점에 메모리로 올라오고, 다시 메모리의 내용이 레지스터로 옮겨져서 해석된다.

E.1.3 x86, x64

인텔의 초기 CPU는 80286, 80386, 80486, 80586(펜티엄)과 같은 식으로 이름이 매겨졌고, 가운데 숫자만 변경해 80x86과 같이 표현할 수 있다. 이를 줄여서 x86이라는 이름이 붙게 됐으며, 대체로 인텔 CPU와 호환되는 제품을 일컬어 x86이라고 한다. 흔히 x86 서버라고 하면 인텔/AMD CPU가 탑재된 서버를 의미한다.

그런데 32비트/64비트를 구분할 때도 x86이라는 용어를 사용한다. 즉, x86은 32비트를, x64는 64비트를 의미하기도 한다.

C#을 이용해 프로그램을 만들 때 EXE/DLL 파일을 x86용(32비트)으로 생성할지, x64용(64비트)으로 생성할지 결정할 수 있다.

E.1.4 멀티 CPU와 멀티 코어

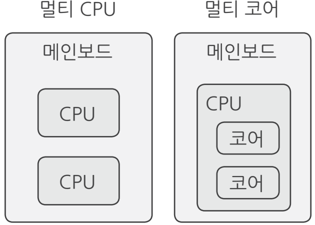
초기 개인용 컴퓨터 시장에서는 컴퓨터 한 대에 한 개의 CPU만 장착했지만 점차 x86 시스템이 서버용 컴퓨터로 사용되면서 2개 이상의 CPU가 장착되는 멀티 CPU를 지원하기 시작했다. 최근에는 CPU의 집적도가 향상되면서 기존의 CPU 자원을 하나의 작은 코어로 만들 수 있게 됐다. 이로써 CPU 한 개에 여러 개의 코어를 집적한 멀티 코어 제품이 등장한다. 우리가 흔히 말하는 듀얼(dual) 코어는 한

개의 CPU에 두 개의 코어를 내장한 것으로, 이는 기존의 CPU 두 개가 했던 일을 하나의 CPU에서 할 수 있게 됐음을 의미한다.

실제로 초기 인텔 제품의 경우 한 개의 CPU 다이(Die)에 두 개의 물리 CPU를 집적시킨 적이 있다.

다음은 멀티 CPU와 멀티 코어의 차이점을 보여준다.

부록 그림 E.1: 멀티 CPU와 멀티 코어



최근에는 서버용 컴퓨터에 16코어까지 탑재된 CPU가 등장하고 있으며, 이 제품을 두 개 장착하면 하나의 컴퓨터에 32개의 코어가 탑재된 시스템이 만들어진다.

CPU/코어가 많으면 동시에 실행할 수 있는 명령어가 증가하므로 전반적으로 시스템 성능이 향상된다.

E.2 운영체제 관련 용어

E.2.1 도스(DOS)

개인용 컴퓨터 시장에서 의미가 있었던 최초의 운영체제는 마이크로소프트에서 만든 MS-DOS(Microsoft Disk Operating System)다. 이것은 16비트 운영체제로 인텔의 8086/8088 CPU에 적합하게 설계됐다. DOS 운영체제는 문자 방식 사용자 인터페이스(CUI: Character User Interface)로도 유명하다. 지금도 DOS의 흔적을 만날 수 있는데, 윈도우의 “명령 프롬프트(Command Prompt)”가 바로 그것이다. 즉, 과거의 DOS 운영체제는 아래의 그림이 모니터 화면 전체를 차지했었다고 보면 된다.

부록 그림 E.2: 명령 프롬프트

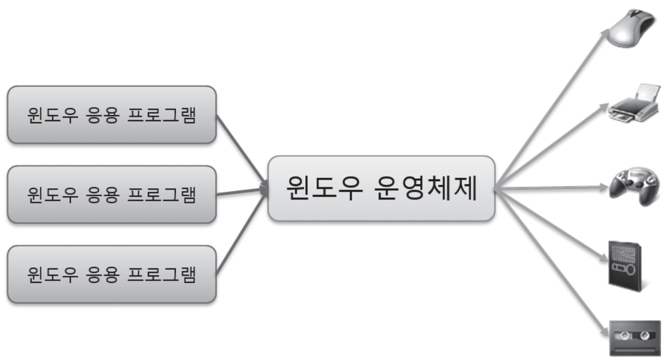
```
C:\Windows\system32\cmd.exe - cmd
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\>
```

윈도우에서 처음 명령 프롬프트를 지원했을 때는 기존 DOS 프로그램과의 호환을 위해서였으나 차츰 유닉스의 셸(Shell)처럼 윈도우에서도 셸의 역할을 담당하게 된다. C#에서도 응용 프로그램을 개발할 때 명령 프롬프트를 이용해 실행하는 프로그램을 만들 수 있다. 이 같은 응용 프로그램 유형을 "콘솔 응용 프로그램(Console Application)"이라 한다.

E.2.2 윈도우 운영체제

마이크로소프트가 MS-DOS에 이어 출시한 그래픽 기반의 사용자 인터페이스(GUI: Graphic User Interface)를 제공하는 운영체제다. DOS에 비해 화려해진 UI와 함께 윈도우는 컴퓨터에 연결되는 모든 장치를 추상화해서 프로그래머에게 공통된 소프트웨어 개발 방식을 제공하는 것이 주요 목표였다.

부록 그림 E.3: 각종 장치를 추상화한 윈도우 운영체제



개발자는 윈도우에서 제공되는 SDK(Software Development Kit)를 이용하면 윈도우 운영체제가 설치된 어떤 컴퓨터에서든 동일한 외형과 기능을 지닌 응용 프로그램을 만들 수 있다.

윈도우는 대표적으로 개인용과 서버용으로 두 가지로 나눠서 출시되고 있다. 개인용을 서버와 대비시켜 클라이언트용이라고도 하며, 윈도우 95, 윈도우 98, 윈도우 ME를 거쳐 윈도우 2000 프로페셔널 버전이 나왔다. 서버용으로는 별도로 윈도우 NT 3.x, 윈도우 NT 4를 거쳐 윈도우 2000 서버 버전이 나온다. 이처럼 윈도우 2000 버전을 시작으로 클라이언트 운영체제가 서버 버전의 커널을 공유하기 시작한다. 이후 서버와 클라이언트 제품이 점차로 비슷한 시기와 사용자 인터페이스를 탑재해서 출시된다. 윈도우 XP와 윈도우 서버 2003, 윈도우 비스타와 윈도우 서버 2008, 윈도우 7과 윈도우 서버 2008 R2로 쌍을 이뤄 각각 클라이언트와 서버 버전이 출시되고, 최근에는 윈도우 8과 윈도우 서버 2012까지 나왔다. 사실상 윈도우 비스타 이후부터는 서버와 클라이언트 버전이 완전히 동일한 소스코드를 공유하고 있다.

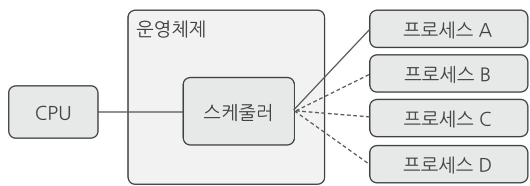
윈도우 7까지는 인텔 호환 CPU만 지원했으나 윈도우 8부터 ARM CPU를 지원하기 시작했다.

E.2.3 멀티 태스킹/ 다중 프로세스

DOS 운영체제에서는 하나의 실행 프로그램이 화면 전체를 차지했다. 따라서 원칙적으로는 한 번에 하나의 프로그램만 실행됐지만 윈도우 운영체제로 오면서 다중 프로세스(Multiple processes) 실행이 가능하게끔 바뀌었다. 즉, 사용자는 화면에 여러 개의 프로그램을 띄울 수 있었고 그 프로그램들은 프로세스(Process)라는 개별 단위로 실행됐다.

초기 개인용 컴퓨터에는 CPU가 하나였다는 점을 상기하자. CPU는 윈도우에 실행된 응용 프로그램을 각각 짧은 시간 동안 실행을 전환시키면서 마치 여러 개의 프로세스가 동작하는 것처럼 구현한다. 예를 들어, 그림 E.4를 보면 현재 A 프로세스에 CPU 자원이 할당돼 있다. 운영체제의 스케줄러는 A 프로세스에 일정 시간 동안 CPU 자원을 할당하고 이후 B 프로세스로 CPU를 할당하고 C, D 프로세스까지 이런 식으로 실행하면서 반복한다.

부록 그림 E.4: 멀티 태스킹



이때 운영체제는 프로세스마다 문맥(Context)이라는 정보를 유지한다. 그리고 스케줄러가 현재 실행 중인 프로세스를 멈추고 다른 프로세스를 실행하는 시점에 문맥 전환(Context Switching)이 발생한다. A 프로세스에서 B 프로세스로 문맥 전환이 발생할 때 스케줄러는 A 프로세스의 실행 정보를 A 프로세스 문맥에 보관하고, B 프로세스의 문맥 정보를 CPU로 불러와 실행한다. 그러다 다시 A 프로세스를 실행하도록 스케줄링되면 운영체제는 기존에 보관된 A 프로세스의 문맥을 CPU로 가져와 이전에 마지막으로 실행했던 명령어 지점부터 다시 실행을 계속한다.

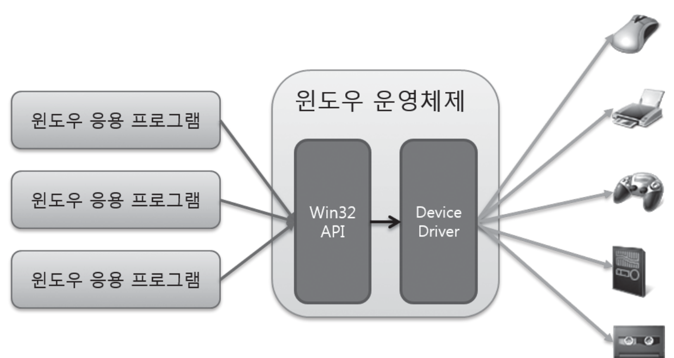
다중 프로세스가 가능하다고 해서 반드시 멀티 태스킹(Multi-tasking)이 되는 것은 아니다. 다중 프로세스는 단지 여러 개의 프로세스가 동시에 운영체제에 의해 로드됐음을 의미할 뿐이다. 만약 운영체제에서 그중 하나만 실행하도록 구현돼 있다면 해당 운영체제는 멀티 태스킹을 지원한다고 볼 수 없다.

E.2.4 Win32 응용 프로그램 인터페이스(API)

윈도우 운영체제에서 동작하는 모든 프로그램은 운영체제 측에서 제공하는 기능을 이용한다. 그 기능을 API(Application Programming Interface)라고 하며, 확장자가 DLL인 파일에서 제공된다.

그림 E.3을 좀 더 구체화하면 다음 그림과 같이 표현할 수 있다.

부록 그림 E.5: Win32 API 공통 인터페이스를 응용 프로그램에 제공



보다시피 윈도우 프로그램은 각종 장치를 직접 제어하지 않는다. 윈도우는 연결될 장치에 대해 그 장치의 사용법을 가장 잘 아는 제조사가 장치 드라이버(Device Driver)를 함께 제공하게 한다. 그리고 API를 통해 장치 드라이버의 기능을 노출한다. 그 덕분에 장치 드라이버가 바뀌었다고 해서 윈도우 응용 프로그램이 바뀌지는 않는다. 이 상황을 "프린터"로 설명해 보자. 프로그램에서는 인쇄를 위해

Win32 API를 사용해 인쇄 작업을 수행할 수 있다. 윈도우는 A 제조사에서 만든 프린터와 장치 드라이버가 있다면 그것을 사용해 인쇄 작업을 연결한다. 또는 B 제조사에서 만든 프린터가 있다면 역시 그 장치 드라이버를 이용해 인쇄 작업을 수행함으로써 윈도우 응용 프로그램과 장치 드라이버 사이의 직접적인 의존성을 제거한다.

윈도우 운영체제가 16비트이던 윈도우 3.1에서는 Win16 API라는 이름으로 불리다가 32비트 운영체제인 윈도우 95가 나오면서 Win32 API라고 불리기 시작했다. 현재 64비트 운영체제까지 나와서 내부적으로 Win64 API가 나오긴 했지만 Win32라는 표현이 워낙 널리 퍼져서 윈도우에서 제공하는 API에 대한 일반적인 호칭으로 "Win32 API"를 주로 사용한다.

처음 Win16 API에서 Win32 API로 넘어올 때는 많은 부분이 바뀌어 이전하기가 쉽지 않았지만 Win32 API에서 Win64로 넘어가는 과정은 잘 만들어진 프로그램의 경우 다시 컴파일하는 것만으로도 가능할 정도로 하위 호환성이 잘 지켜진다.

Win32 API를 호출해서 동작하는 응용 프로그램은 일반적으로 모든 종류의 윈도우에서 실행된다. 단지 새로운 운영체제에서는 이전 운영체제에서 제공되지 않는 새로운 API를 제공하고, 서버 운영체제는 클라이언트 운영체제가 제공하는 것보다 더 많은 API를 제공하기도 한다.

C# 응용 프로그램에서는 Win32 API를 직접 호출할 수 있지만 이렇게 만든 프로그램은 리눅스에서 실행할 수 없다. 따라서 특별한 목적이 없다면 사용하지 않기를 권장한다.

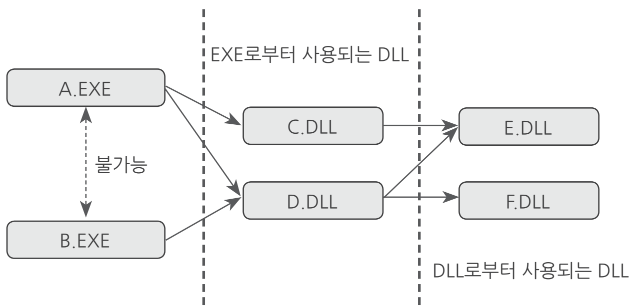
E.2.5 윈도우 응용 프로그램

윈도우 운영체제에서 실행되는 프로그램을 말한다. 모든 윈도우 응용 프로그램은 Win32 API를 사용해 운영체제에서 제공하는 기능들과 협력한다. 윈도우에서는 Win32 API를 동적 링크 라이브러리(DLL: Dynamic Link Library) 파일로 제공한다. 탐색기를 열어 C:\Windows\System32 폴더를 보면 무수히 많은 DLL 파일이 있는 것을 확인할 수 있는데, 모두 윈도우 운영체제에서 제공하는 기능에 해당한다.

응용 프로그램 개발자가 직접 사용하는 기본 DLL은 kernel32.dll, user32.dll, gdi32.dll 정도가 있다. Visual C++ 언어를 사용하는 윈도우 프로그래머는 최소한 그 3개의 DLL에서 제공하는 API의 사용법에 익숙해져야 한다.

윈도우에서 사용자가 실행하는 파일은 보통 확장자가 EXE인데, EXE 파일은 DLL 파일과 구조가 동일하다. 다만 DLL은 내부에 구현된 API를 외부에서 참조할 수 있다. 따라서 EXE 파일은 다른 EXE 파일의 기능을 사용할 수 없지만 DLL 파일은 로드해서 사용할 수 있다.

부록 그림 E.6: DLL과 EXE의 참조 관계



이러한 이유로 재사용 가능한 코드는 컴파일해서 DLL 파일에 모아두고 서로 다른 EXE에서 공유해서 사용하는 방식이 일반적이다. 예를 들어, 파일을 암호화/복호화하는 코드를 작성했다고 가정해 보자. 그 코드를 DLL 파일에 넣어두면 A.EXE, B.EXE에서는 그 DLL을 로드해서 암호화/복호화하는 API를 호출해서 사용할 수 있다. 반면 DLL이 아닌 EXE 파일에 넣었다면 그와 같이 재사용하는 것이 불가능하다.

대부분의 윈도우 응용 프로그램은 1개의 EXE 파일과 여러 개의 DLL 파일로 구성된다. 이는 C#으로 프로그램을 만들 때도 마찬가지다.

E.2.6 32비트 응용 프로그램

32비트 CPU의 기계어로 번역된 프로그램을 일컫는다. 32비트 응용 프로그램의 특징은 해당 프로그램이 사용할 수 있는 메모리의 주소가 2^{32}(4GB)라는 제약이 있다. 4GB라는 용량이 크다고 느낄 수도 있지만 최근의 서버용 응용 프로그램에서는 4GB로도 모자라는 경우가 종종 발생한다. 메모리 용량뿐 아니라 CPU와 메모리 간의 데이터 이동이 32비트 단위로 발생하고 CPU 내의 레지스터 크기가 32비트이므로 한 번에 32비트 데이터만큼의 계산만 할 수 있다.

64비트 윈도우의 경우 32비트에 대한 호환성을 유지하고 있어 별다른 변경 없이 기존의 32비트 응용 프로그램을 그대로 실행할 수 있다. 대신 64비트의 혜택을 고스란히 제공받지 못하고 32비트의 제약을 그대로 유지한 채 실행된다. 게다가 32비트 응용 프로그램을 64비트 환경에서 실행하기 위해 중간

에 변환을 위한 층을 하나 두는데, 이를 WoW64(Windows 32bit on Windows 64bit)라고 한다. 따라서 32비트 응용 프로그램을 32비트 운영체제에서 실행하는 것보다 64비트 운영체제에서 실행했을 때 속도가 더 느려진다는 단점이 있다.

32비트 윈도우 운영체제는 4GB 가상 메모리 중 2GB 영역을 운영체제 전용으로 예약하기 때문에 사실상 응용 프로그램에서 사용할 수 있는 메모리 주소는 2GB에 불과하다.

E.2.7 64비트 응용 프로그램

64비트 운영체제에서만 실행되는 응용 프로그램이다. 마이크로소프트에서는 서버 운영체제의 경우 Windows 서버 2008 R2 제품 이후로 64비트 운영체제만 출시하고 있으므로 서버용 응용 프로그램에 관심이 있다면 64비트 환경에 익숙해지는 것이 좋다.

64비트는 메모리 주소를 2^{64}로 지정할 수 있는데, 이론상 16EB(Exabyte)의 메모리를 다룰 수 있지만 현실적인 이유로 윈도우에서는 커널 모드의 주소 공간으로 8TB(Terabyte)를, 사용자 모드의 주소 공간으로 8TB를 각각 예약한다.

CPU와 메모리 간의 데이터 이동이 64비트 단위로 발생하고 CPU 내의 레지스터 크기가 64비트이므로 한 번에 64비트 데이터만큼 계산할 수 있다. 이로 인해 경우에 따라 32비트 응용 프로그램과 비교해 2배 가까운 속도 향상이 가능해진다.

32비트 실행 파일과 64비트 실행 파일은 동일한 프로세스에 로드될 수 없다. 예를 들어, 32비트로 컴파일된 DLL은 64비트로 컴파일된 EXE 또는 DLL과 함께 사용할 수 없으며, 그 반대도 마찬가지다.

C#을 이용하면 32비트와 64비트 응용 프로그램을 모두 만들 수 있다. 또한 “AnyCPU”라는 모드가 추가돼 있는데, 이 모드로 만들어진 C# 응용 프로그램은 32비트 운영체제에서는 32비트로, 64비트 운영체제에서는 64비트로 실행된다는 특징이 있다.

E.2.8 윈도우 이외의 운영체제

마이크로소프트 윈도우 운영체제 외에도 개발자로서 주목해야 할 운영체제들이 있다.

- 유닉스(Unix)

  특정 분야를 제외하고는 현재 유닉스가 그렇게 중요한 위치를 차지하고 있지는 않지만, 현대 운영체제의 개념을 확립했다는 점에서 의미가 있다.

- 리눅스(Linux)

  유닉스에 뿌리를 두고 있지만 개인이 만들어 오픈소스로 공개한 운영체제다. 무료로 공개된 운영체제라는 특징으로 인해 클라이언트 및 서버 운영체제에 걸쳐 폭넓게 사용되고 있다.

- 맥 OS X

  애플의 맥 하드웨어를 기반으로 실행되는 운영체제로서 GUI 기반이지만 유닉스에 뿌리를 두고 있어 터미널 모드에서 사용되는 명령어는 유닉스/리눅스와 호환되는 부분이 많다.

그리고 다음의 운영체제들은 보통 모바일 기기에 설치된다.

- iOS

  애플의 휴대전화 및 태블릿 제품인 아이폰(iPhone), 아이패드(iPad)에 탑재된다.

- 안드로이드

  리눅스를 모바일 환경에 최적화한 운영체제다. 구글에서 개발했고 무료로 배포하기 때문에 각종 모바일 기기에 사용되고 있다.

- 윈도우 폰 OS

  윈도우 역시 핸드폰을 위한 경량화된 운영체제를 개발했고, 현재 윈도우 폰 8까지 출시됐다.

컴퓨터에 사용되는 운영체제는 대체로 x86 호환 시스템에서 운영되는 반면, 모바일/임베디드 기기에 사용되는 운영체제는 대부분 ARM 계열의 CPU에서 동작한다.

E.3 프로그래밍 용어

E.3.1 기계어

기계어(Machine language)란 CPU가 해석할 수 있는 2진수의 모음이다. 예를 들어, x86 CPU에서 EAX라고 불리는 레지스터의 값을 1과 비교하는 연산에 해당하는 기계어는 "1000 0011 1111 1000 0000 0001"이고 16진수로 표현하면 "83 F8 01"에 해당한다. 기계어도 하나의 프로그래밍 언어다. 단지 특별한 문법이 없고 그 기계어가 실행되는 CPU가 정한 규칙에 따라 2진 숫자를 나열해야 한다.

지금 세대는 잘 믿기지 않겠지만 과거에는 이처럼 숫자로 된 기계어로 프로그램을 만들던 적도 있었다.

E.3.2 어셈블리어, 소스코드, 컴파일

숫자로만 이뤄진 기계어로 프로그램을 만들기란 너무 불편하다. 이러한 기계어에 의미를 부여할 수는 없을까? EAX 레지스터와 1을 비교한다는 작업을 숫자가 아닌 문자열로 "CMP EAX, 1"과 같이 표현하면 어떨까?

```
기계어: 1000 0011 1111 1000 0000 0001
→ Compare EAX and 1
→ CMP EAX, 1
```

보다시피 훨씬 이해하기 쉬워졌다. 이처럼 기계어로 하는 작업에 문자열로 의미를 붙여 만들어진 언어가 바로 어셈블리어(Assembly Language)다. 이 언어는 기계어와 거의 일대일로 대응된다.

여기서 "CMP EAX, 1"은 어셈블리어로 작성한 소스코드(Source code)에 해당한다. 물론 CPU는 소스코드를 직접 이해할 수 없으므로 그에 대응되는 기계어인 1000 0011 1111 1000 0000 0001로 바꾸는 작업이 필요하다. 이 과정을 컴파일(Compile)이라고 하며, 그와 같은 작업을 하는 프로그램을 컴파일러(Compiler)라고 한다. 컴파일러 중에서도 어셈블리 언어의 컴파일러를 특별히 어셈블러(Assembler)라고 한다.

E.3.3 컴파일러, 링커, 빌드

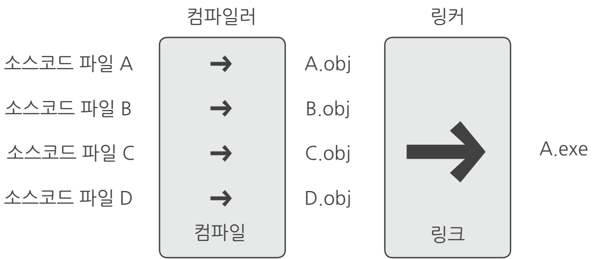
프로그래밍 언어 가운데 컴파일러(Compiler)를 제공하는 것이 많다. 하지만 CPU는 그러한 언어의 고유 문법에 맞춰 작성된 소스코드를 이해할 수 없기 때문에 반드시 기계어로 바꿔야 한다. 이 과정을 좀 더 자세하게 설명해 보자. 관례상 어셈블리어로 소스코드를 작성하면 확장자가 asm으로 된 파일에 저장한다. 어셈블러로 asm 파일을 컴파일하면 확장자가 obj인 파일이 생성된다. 그런데 왜 exe가 아닌 obj일까?

프로그램을 한 개의 소스코드 파일로 만드는 것은 대단히 불편한 일이다. 물론 규모가 작은 프로그램이라면 한 개의 파일로도 충분하겠지만 대부분의 의미 있는 프로그램들은 여러 개의 파일로 나눠서 작성한다. 따라서 한 개의 asm 파일에 대해 한 개의 exe 파일로 출력할 수는 없다. 하나의 프로그램을

구성하기 위해 만들어진 모든 소스코드 파일을 컴파일하면 각각에 해당하는 obj 파일이 나오게 하고, 그 obj 파일들을 모아서 하나의 exe로 만드는 작업을 하게 된다.

앞에서 설명한 내용을 토대로 다시 컴파일러를 정의해 보자. 컴파일러란 소스코드에서 그것이 의미하는 기계어로 변환해 obj 파일에 담는 역할을 한다. 그리고 obj 파일을 모아서 하나의 exe 파일을 만드는 작업을 링크(Link)라고 하며, 그 작업을 수행하는 프로그램을 링커(Linker)라고 한다.

부록 그림 E.7: 컴파일러와 링커의 역할



프로그램을 하나 만드는 과정은 컴파일 과정과 링크 과정으로 분리돼 있지만 일반적으로 "컴파일한다"라는 표현에는 그 두 가지 과정이 모두 포함된다. 최근에는 "컴파일 + 링크" 과정을 합쳐 '빌드(build)'라는 표현이 많이 쓰이는 추세다.

E.3.4 인터프리터 언어

어셈블리어는 컴파일을 통해 소스코드를 중간 파일(obj)로 출력하고 다시 링크 과정을 거쳐 실행 가능한 파일로 출력한다. 이러한 언어를 컴파일러 언어라고 한다. 반면 인터프리터 언어(Interpreter Language)는 명시적인 컴파일 과정 없이 소스코드에서 곧바로 프로그램을 실행하는 기능을 제공한다.

대표적인 예로 자바스크립트(JavaScript)가 있다. 자바스크립트는 실행을 위해 별도로 컴파일러를 사용하지 않는다. 단지 HTML 페이지 안에 자바스크립트 코드를 넣어두면 웹 브라우저가 소스코드의 문장을 하나씩 실행한다.

인터프리터 언어는 실행 시점에 소스코드를 해석한다는 특징이 있다. 반면 컴파일러 언어는 소스코드를 컴파일 과정에서 실행 가능한 기계어로 번역해 파일로 저장해 둔다. 이 같은 차이점으로 인해 일반적으로 실행 속도는 인터프리터 언어보다 컴파일러 언어가 더 빠르다.

C#은 컴파일러 언어라서 실행하기 전에 반드시 빌드 과정을 통해 결과 파일을 만들어야 한다.

E.3.5 저급/고급 프로그래밍 언어

기계어와 어셈블리어를 저급언어(Low Level Language)라고 한다. 비록 어셈블리어가 기계어보다 쉬워진 것은 사실이지만 개발자는 좀 더 편한 구문을 원했다. 예를 들어, "CMP EAX, 1" 같은 식의 문법을 좀 더 추상화해서 다음과 같이 표현하는 것도 가능하다.

```
if (n == 1)
{
   n = n * 2;
}
```

어셈블리어를 기계어로 변환한 것처럼, 위의 구문도 적절하게 기계어로 변환하는 또 다른 컴파일러를 제공할 수 있다면 가능한 일이다. 이렇게 탄생한 언어가 바로 고급 프로그래밍 언어(High Level Language)다. 초기의 포트란(Fortran), 코볼(Cobol), 파스칼(Pascal), C 언어와 같은 컴파일러 언어와 BASIC, LISP 같은 인터프리터 언어가 모두 고급 언어에 속하며, 이후에 나온 프로그래밍 문법을 가진 모든 언어도 여기에 속한다. C# 역시 고급 언어의 하나다.

E.3.6 네이티브 언어

컴파일러가 출력한 결과물이 특정 CPU를 위한 기계어일 때 이 언어를 네이티브 언어(Native Language)라고 한다. 어셈블리어는 당연히 네이티브 언어이고, 이후의 초기 언어는 대부분 여기에 속한다. 일반적으로 네이티브 언어로 만들어진 실행 파일은 속도가 빠르다는 특징이 있다. 왜냐하면 출력 결과물이 기계어라서 CPU에 의해 곧바로 해석될 수 있기 때문이다. 이 같은 특징은 한편으로는 단점으로 작용할 수 있는데, 가령 x86용 실행 파일은 ARM CPU에서 실행할 수 없다.

대표적인 네이티브 언어로 C, C++ 언어가 있다. 마이크로소프트의 경우 네이티브 언어라는 말과 함께 비관리 언어(Unmanaged Language)라는 말도 함께 쓰고 있다.

E.3.7 프로세스 가상 머신(VM)

자바와 닷넷 프레임워크의 공통점은 프로세스 가상 머신(Virtual Machine)에 있다. 그러한 환경의 프로그램을 실행하면 프로세스 내에 작은 가상 머신이 생성된다. 가상 머신은 말 그대로 "가상의 기계" 역할을 한다.

일반적으로 VM에서 해석하는 기계어는 그 VM이 실행되는 컴퓨터의 CPU에서 해석되는 기계어와는 다르다. 자바의 경우 자바 가상 머신에서 해석되는 기계어를 바이트 코드(Byte Code)라 하고, 닷넷 프레임워크에서는 중간 언어(IL: Intermediate Language) 코드라고 한다.

보통은 가상 머신의 고유한 기계어를 모두 중간 언어라고 통칭한다.

중간 언어는 "공통 기계어"로서의 역할을 수행한다. 개발자가 작성한 VM용 프로그램은 중간 언어로 된 파일로 만들어진다. 기존의 네이티브 언어로 된 프로그램이 특정 CPU에 종속되는 기계어를 생성한 것과 대비된다. 물론 VM의 중간 언어는 CPU가 곧바로 실행할 수 없다. 이러한 이유로 VM은 중간 언어를 CPU가 실행할 수 있는 기계어로 변환하는 작업도 담당한다.

VM 환경의 또 다른 특징은 "공통 실행 파일 포맷"에 있다. 윈도우에서 C/C++ 언어의 소스코드가 컴파일되면 윈도우 운영체제가 자체적으로 정의한 실행 파일 포맷에 맞게 기계어를 담는다. 마찬가지로 리눅스는 나름대로의 실행 파일을 정의하고 있으며, 이는 윈도우의 실행 파일 포맷과 완전히 다르다. 따라서 VM에서 정의된 공통 실행 파일 포맷에 따라 컴파일러들이 실행 파일(예: EXE 또는 DLL)을 생성해 준다면 VM은 그 파일을 손쉽게 메모리로 읽어와 실행할 수 있다.

기존에는 개발자가 만든 실행 프로그램이 해당 환경에 종속됐다. 예를 들어, 윈도우 환경에서 만든 프로그램은 오직 윈도우에서만 실행됐다. 하지만 중간 언어를 다루는 VM이 제공되면서 상황이 완전히 바뀐다.

부록 그림 E.8: VM의 역할

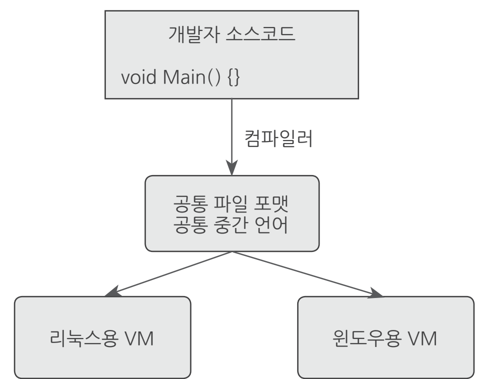


그림 E.8처럼 윈도우나 리눅스, 심지어 ARM CPU 기반의 운영체제에서도 중간 언어를 해석할 수 있는 VM만 제공된다면 개발자가 만든 프로그램을 실행할 수 있게 된 것이다.

VM의 대표적인 사례가 바로 자바의 JVM(Java Virtual Machine)과 닷넷 프레임워크의 CLR(Common Language Runtime)이다.

E.3.8 가상 머신 지원 언어

이를테면, 자바와 C#이 대표적인 언어에 해당한다. 이러한 언어로 작성된 소스코드를 컴파일하면 기계어로 된 결과물을 산출하지 않는다. 대신 각각 JVM과 CLR에서 지원되는 중간 언어로 결과물을 생성한다. 이러한 출력물은 일반적인 기계어를 담고 있지 않으므로 직접 실행될 수 없고 반드시 가상 머신(JVM 또는 CLR) 내에서만 동작할 수 있다.

비교를 위해 기존의 네이티브 언어로 만들어진 결과물의 특징을 살펴보자.

- 네이티브 언어로 컴파일된 결과물은 그것이 실행될 CPU의 기계어로 돼 있다.
- 네이티브 언어로 컴파일된 결과물은 그것이 실행될 운영체제에서 제공되는 시스템 함수를 사용한다(윈도우의 경우 Win32 API를 사용해서 제작되지만 리눅스에는 Win32 API가 없으므로 실행할 수 없다).
- 네이티브 언어로 컴파일된 결과물은 그것이 실행될 운영체제에서 요구하는 파일 포맷으로 돼 있다(윈도우와 리눅스의 실행 파일 포맷은 다르다).

따라서 네이티브 언어로 컴파일된 실행 파일은 CPU 및 운영체제가 다른 환경에서는 정상적으로 실행되지 않는다. 이러한 제약은 중간 언어를 둔 프로세스 가상 머신으로 해결할 수 있다.

가상 머신을 지원하는 언어는 단지 공통된 실행 파일 포맷에 미리 약속된 중간 언어로 된 결과물을 만들어낸다. 그리고 각 운영체제에서는 그러한 중간 언어를 실행 가능하게끔 번역하는 프로세스 가상 머신을 만들면 된다. 따라서 자바와 C#으로 만든 응용 프로그램은 대상이 되는 환경에 적합한 프로세스 가상 머신(JVM 또는 CLR) 환경을 미리 설치해 둬야 한다.

참고 자료

- 마이크로소프트 개발자 웹 사이트
  – http://msdn.microsoft.com
- 저자의 웹 사이트
  – http://www.sysnet.pe.kr
- 마이크로소프트 사내 개발자들의 블로그
  – http://blogs.msdn.com/
- MSDN Magazine
  – http://msdn.microsoft.com/en-us/magazine/default.aspx
- 마이크로소프트웨어: http://www.imaso.co.kr/
- CLR via C# 2nd Edition 한국어판 (송기수 역, 정보문화사, 2008년 11월 20일 출간)
- The C# Programming Language (Fourth Edition) 한국어판 (김도균, 안철진 역, 에이콘출판사, 2012년 6월 29일 출간)
- C# 언어 명세: 비주얼 스튜디오 2012에 제공되는 CSharp Language Specification.docx
- 닷넷의 가비지 컬렉션: http://msdn.microsoft.com/en-us/magazine/bb985010.aspx
- C# 오픈소스 컴파일러 – Roslyn
  https://github.com/dotnet/roslyn

[기호]

^ 282
-- 93
-= 95, 202
!= 83
* 289
*= 95
/= 95
& 282, 289
%= 95
++ 93
+= 95, 202
〈 83
〈= 83
== 83
=〉 619
〉 83
〉= 83
| 282
~ 282
0.0.0.0 477
0세대 354
1급 함수 200
1세대 354
2세대 355
2의 보수 284
2진 데이터 395
3항 연산자 88
32비트 기본 사용 320
127.0.0.1 480
&&(AND: 논리곱) 83
/checked 285
/debug 313
/debug:full 314
/debug:pdbonly 314
/define 267
#define 269
#elif 269
#else 269
#endif 267
%ERRORLEVEL% 133
__FILE__ 662
/flushdns 471
#if 267
/keyfile 323
__LINE__ 662
!(NOT: 부정) 83
||(OR: 논리합) 83
/out 297
#pragma 709
/r 299
/target:exe 299
/target:library 299
#undef 269
/unsafe 290
^(XOR: 배타적 논리합) 83

[ㄱ – ㄹ]

가변 객체 248
가변 배열 81
가비지 수집기 71, 127, 352
가상 메서드 177
간편 표기법 148
값에 의한 전달 234
값에 의한 호출 234
값 형식 70, 71
강력한 결합 225, 567
강력한 이름의 어셈블리 323
강력한 이름 키 파일 선택 324
개인(private, 사설) IP 464
객체 110, 127
객체 지향 111
객체 지향 프로그래밍 111
객체 지향 프로그래밍 언어 111
객체 초기화 609
계승 350
계약 213, 218
계층(tier) 533
계층 상속 152
고립성(Isolation) 548
공개키 기반 구조 322
공개키 토큰값 322
공용(public) IP 464
공용 속성 146
공용 언어 기반구조 32
공용 언어 런타임 33
공용 언어 사양 29
공용 타입 시스템 28
공유 리소스 442

관계 연산자 83
관계형 데이터베이스 499
관리 코드 288
관리 프로세스 34
관리 환경 34
관리 힙 127, 352
구성(Configuration) 옵션 314
구조체 227
구현 타입 227
국제 인터넷 번호관리기관 464
그리니치 평균시 375
기반(base) 클래스 149
기본값 74
기본 경로 420
기본 문서 492
기본 생성자 125
기본 응용 프로그램 도메인 556
기본 자료형 56
기본 키 515
깊은 복사 230
%(나머지 연산자) 76
날짜 분기선 375
내림차순 206
내부 조인 640
네임스페이스 137
네트워크 어댑터 463
논리 연산자 83
논블로킹 호출 457
느슨한 결합 225, 567
다중 상속 152, 213
다중 소스코드 파일 296
다차원 배열 80
다형성 174, 216
단락 계산 85
단순 대입 연산자 95
단일 상속 152
단축 평가 85
단항 188
닫힘 태그 294
닷넷 4.0의 교체판 661
닷넷 역컴파일러(Decompiler) 52
닷넷 프레임워크 25, 34
닷넷 프레임워크 2.0 575
닷넷 프레임워크 3.0 575
닷넷 프레임워크 3.5 604
닷넷 프레임워크 4.0 649
닷넷 프레임워크 4.6 680
닷넷 프레임워크는 4.5 661
닷넷 호환 언어 26
대상 프레임워크 305
대용량 객체 힙 359
대입 연산자 76
덕 타이핑 657
데이터그램 소켓 472
데이터로서의 람다 식 627
데이터 컨테이너 529
델리게이트 196
도메인 네임 서버 463, 469
도메인 이름 463
동기 호출 455
동기화 442
동적 언어 652
동적 타입 언어 64
디버거 311
디버그 빌드 314
디버깅 311
디버깅 시작 301
디버깅하지 않고 시작 301
라이브러리 298
라이브러리 패키지 관리자 657
람다 식 618
람다 식을 이용한 메서드, 속성 및 인덱서 정의 683
레이블 문 104
레지스트리 편집기 568
레코드 499
루트 참조 358
루프(loop)문 92
루프백 호스트명 496
름 공간 137
리터럴 68
리틀 엔디안 394
리플렉션 551
릴리즈(release) 빌드 313
마이크로소프트 SQL 서버 499
매개변수 116, 117
매개변수화된 쿼리 526
매니페스트 31
매크로(macro) 상수 662
메모리 누수 현상 352
메모리 파편화 359
메서드 115
메서드 시그니처 183

메서드 오버라이드 174, 178
메서드 오버로드 184
메타데이터 30, 551
멤버 메서드 122
멤버 변수 115
명명된 인자 651
명시적 변환 66
명시적 형변환 154
모노(Mono) 프로젝트 43
모듈 31
무한 집합 593
문맥 예약어 263
문자열 보간 693
문자형 기본 타입 60
문장부호 75

[ㅂ – ㅇ]

바이트 순서 394
바이트 코드 27
바인딩 475
박싱 352
박싱/언박싱 575
반복문 92
배경 스레드 433
배열 76
버그 311
버전 320
범용 데이터 컨테이너 534
변수 57
변수(variable) 69
변수를 선언 57
병렬 처리 라이브러리 671
복합 대입 연산자 94
부모 클래스 149
부분 메서드(partial method) 613
부울 대수 83
부하 분산 470
부호화 388
불변 객체 248, 386
블랙박스 121
블로킹 호출 455
블록 86
비관리 메모리 364
비관리 코드 288
비동기 통신 487
비동기 호출 455, 663
비주얼 스튜디오 46
비주얼 스튜디오 2005 575
비주얼 스튜디오 2008 604
비주얼 스튜디오 2010 649
비주얼 스튜디오 2012와 2013 661
비주얼 스튜디오 2015 680
비주얼 스튜디오 2015 다운로드 47
비트 논리 연산자 282
빅 엔디안 394
빅–오 표기법 411
빈 솔루션 295
사용자 계정 컨트롤 570
사용자 정의 예외 타입 341
산술 연산 오버플로/언더플로 확인 285
산술 연산자 76
상속 148
상수 74, 249
상수식 75
상호운영성 403
생성자 122
서명 323
서브(sub) 클래스 149
서브루틴 115
선입선출 415
선입후출 413
선택문 82
선택적 매개변수 650
선택 정렬 204
설정자 메서드 144
세계 표준시 375
세대 354
소멸자 126, 364
소수(Prime) 생성기 252
소스코드 40
소켓 472
속성 110, 146
솔루션 294
솔루션 탐색기 293
수동 리셋(manual reset) 이벤트 454
순서도 107
슈퍼(super) 클래스 149
스레드 428
스레드 문맥 429
스레드 문맥 전환 489
스레드 안전성 447

스레드에 안전하지 않은(not thread-safe) 메서드 446
스레드에 안전한(thread-safe) 메서드 446
스레드 풀 450
스택 70, 71, 346
스택 오버플로 349
스택 트레이스 337
스트림 소켓 472
시간대 375
시작 프로젝트 301
시작 프로젝트로 설정 301
시프트 연산자 280
식별자(identifier) 67
식 트리 627
실수형 기본 타입 59
실체화된 타입 227
실행 계획 526
실행 시 오류 311
싱글턴 131
안전하지 않은 컨텍스트 289
안전하지 않은 코드 허용 290
알고리즘 106
암시적 변환 65
암시적 형변환 154, 191
액세스 포인트 465
얕은 복사 230
어셈블리 31
어셈블리 서명 324
어셈블리 이름 322
언더플로 284
언박싱 352
엔티티 프레임워크 547
역직렬화 393
연결 문자열 518
?? 연산자 590
연산자 75
연산자 오버로드 186
연산자 우선순위 283
열거자 221
열거형 242
열림 태그 294
예약어 67
예외 329
예외를 먹는(swallowing exceptions) 상황 344
예외 처리기 333
예외 필터 699
오류 329
오름차순 206
오버플로 284
와일드카드 문자 425
외부 조인 640
요청 492
원자성(Atomicity) 548
원자적 연산 448
웹 서버 494
유니코드 61
유효 범위 270
응답 492
응용 프로그램 구성 파일 303
응용 프로그램 도메인 551
이름 충돌 137
이벤트 251
이스케이프 시퀀스 61
이항 188
익명 메서드 600, 619
익명 타입 613
인(private) 어셈블리 325
인덱서 259
인덱스 499, 514
인라인 313
인스턴스 111, 127
인스턴스 메서드 127
인스턴스 멤버 127
인스턴스 생성자 127
인스턴스 필드 127
인자 117
인코딩 388
인터넷 서비스 업체 465
인터페이스 213
일관성(Consistency) 548
읽기 전용 필드 247

[ㅈ – ㅎ]

자동 구현 속성 606
자동 구현 속성 초기화 681
자동 리셋(auto reset) 이벤트 454
자료형 56
자손(descendant) 클래스 149
자식 클래스 149
자식 태그 294
작업 관리자 428
재귀 호출 350

전경 스레드 433
전역 어셈블리 328
전역 어셈블리 캐시 326
전용 어셈블리 325
전위 표기법 93
전처리기 지시문 266
전체 가비지 수집 359
점프문 102
접근자 메서드 144
접근 제한자 142
접점 468
정규 표현식 389
정보 은닉 143
정수형 기본 타입 56
정수형 포인터 571
정적 메서드 131
정적 멤버 128
정적 생성자 134
정적 클래스 602
정적 타입 언어 64
정적 필드 129, 558
제네릭 575
제네릭 메서드 581
제네릭 클래스 580
제어문 82
조건부 컴파일 기호 268
조건 연산자 88
조상(ancestor) 클래스 149
종단점 468
종료 큐 368
종점 468
주 스레드 429
중첩 루프 98
중첩 클래스 192
증감 연산자 93
지속성(Durability) 549
지역 변수 115
지역 시간 376
지연된 연산 646
지연된 평가 625
지연 실행 646
직렬화 393
진입점 133
참조 299
참조 관리자 301
참조에 의한 전달 234
참조에 의한 호출 234
참조형 113
참조 형식 70, 71
처리되지 않은 예외 333
첨자 78
최상위 비트 281
최적화 312
추상 메서드 193
추상 클래스 193
추상화 122
층(Layer) 533
캐스팅(casting) 66
캡슐화 141
커서 524
컨테이너 508
컬렉션 407
컬렉션 초기화 610
컬렉션 초기화 구문에 확장 메서드로 정의한 Add 지원 707
컴파일 40
컴파일러 41
컴파일 시 오류 311
코드로서의 람다 식 618
코드 분석기 172
콘솔 응용 프로그램 294
콜백 메서드 203
클라이언트/서버 466
키워드 67
타입 111
타입 추론 604
태그 294
테이블 507
통합 개발 환경 44
트랜잭션 548
특성 245, 271
틀(frame) 111
파생(derived) 클래스 149
파워셸 34
파일 참조 301
파일 탐색기에서 폴더 열기 293
패키지 관리자 콘솔 657
팩터리 메서드 629
포인터 289
포트 466
표준 쿼리 연산자 641
프로시저 115
프로젝트 293

프로젝트 오일러 106
프로젝트 참조 301
프로토콜 463
프로퍼티 146
플랫폼 대상 319
플랫폼 호출 288
플러그인 563
피호출자 203
필드 114, 146
필수 매개변수 651
하위 호환성 306
한국 표준시 375
함수 115
합성 함수 119
해시 충돌 165
핸들(HANDLE) 571
행위 111
협정 세계시 375
형변환 65, 152
형변환 연산자 66
형식 매개변수 580
형식 매개변수에 대한 제약 조건 582
형식 문자열 382
형식 안전성 540
형식 이니셜라이저 134
호출 스택 337
호출자 203
호출자 정보 662
혼합 모드 503
확장 메서드 615
확장 모듈 구현 563
후위 표기법 93
힙 70, 71, 352

[A – D]

abstract 194
Accept 484
ACID 548
Action 620
Activator.CreateInstance 559
ActiveX 320
AddressFamily 469, 472
AddressList 469
ADO.NET 데이터 제공자 517
AllocCoTaskMem 365
AllowMultiple 277
AnyCPU 319
app.config 303
AppDomain 551
AppDomain.CreateDomain 556
AppDomain.CurrentDomain 552
AppDomain.Unload 558
Append 387
appSettings 308, 309
ArgumentException 408, 412
ARM 32비트 320
ArrayList 575
ArrayList.Sort 409
Array.Sort 219
as 155
ascending 636
ASCII 388, 389
AssemblyCopyright 279
AssemblyFileVersion 279
AssemblyInfo.cs 278, 321
AssemblyInformationalVersion 279
Assembly.LoadFrom 562
AssemblyProduct 279
AssemblyTitle 279
AssemblyVersion 321
async 663
await 663
await Task.WhenAll 679
base 172, 179
BCL 34, 298, 372
BeginInvoke 460, 674
BeginRead 456
BeginTransaction 549
BeginWrite 456
bigint 509
BigInteger 570
BigInteger.Parse 571
BinaryExpression 628
BinaryWriter 418
BlockExpression 629
BOM 400
bool 64
bool? 599
break 90, 97, 102

byte 58
C# 2.0 575
C# 3.0 604
C# 4.0 649
C# 5.0 661
C# 6.0 680
CallerFilePath 663
CallerLineNumber 663
CallerMemberName 663
CallSite 654
CallSite.Target 654
case 89
catch 332
CatchBlock 629
catch/finally 블록 내에서 await 사용 가능 706
CBV 234
cctor 135
ChangeExtension 426
char 60
checked 284
CIL 27
class 112
ClassesRoot 569
Class Library 300
CLI 32
Client Profile 305
CLR 2.0 304
CLR 4.0 304
CLS 29
Commit 549
Compile 628
Concrete 타입 227
ConditionalExpression 629
Conditional 특성 316
Configuration.ConfigurationManager.AppSettings 309
ConfigurationSettings 309
ConnectionString 520
Console Application 300
const 74
ConstantExpression 629
constructor 123
Contains 379
continue 102
Copy 166
CP949 388
CreateDirectory 423
CreateInstanceFrom 556
CRUD 510
C/S 466
CS0618 309
csc.exe 41, 299
csc.rsp 299
csproj 293
ctor 123
CTS 28
CurrentConfig 569
CurrentUser 569
DAC(Data Access Component) 532
Data Source 519
DateTime.Now 374
DateTime.UtcNow 376
Debug 316
DEBUG 315
DebugInfoExpression 629
DebugView 318
decimal 59
default 89, 592
Default 389
DefaultExpression 629
DefaultIfEmpty 640
default(T) 591
delegate 196, 601
DELETE 516
Dequeue 415
descending 636
destructor 126
Dictionary.Add 698
Dictionary<TKey 589
Dictionary 타입의 인덱스 초기화 698
Directory.EnumerateFiles 564
Directory.GetDirectories 424
Directory.GetFiles 424
Directory.GetLogicalDrives 424
DirectoryNotFoundException 421, 423
Dispose 361
DLL 298
DllImport 288
DLL 지옥 321
DLR 652
dmcs 43
Dns.GetHostEntry 468
DOS 416

double 59
double? 600
do/while 문 101
DownloadFile 498
DownloadStringAsync 669
DownloadStringTaskAsync 669
DreamSpark 47
dtor 126
dynamic 652
DynamicExpression 629

[E – K]

ElapsedTicks 378
ElementInit 629
else 87
Encoding 388
EndInvoke 460, 675
EndRead 456
EndsWith 379
EndWrite 456
Enqueue 415
Enumerable 616
Environment.CurrentDirectory 420
Environment.Is64BitProcess 319
Environment.NewLine 400
Environment.Version 307
Epoch Time 377
Equals 162, 181
EUC-KR 388
event 251
ExecuteNonQuery 521
ExecuteReader 521
ExecuteScalar 521
explicit 191
extern 287
F5 단축키 317
factorial 350
FCL 298
FileAccess 418
FileAccess.ReadWrite 419
FileAccess.Write 418
File.Copy 421
File.Delete 422
File.Exists 422
FileMode 418
FileMode.Append 419
FileMode.Create 419
FileMode.OpenOrCreate 419
File.Move 422
FileNotFoundException 418
File.ReadAllText 422, 674
FileShare 418
FileShare.None 419
FileStream 361
File.WriteAllText 422
Fill 528
finally 334
FindAll 624
fixed 290
Flags 245
float 59
Flush 398
foreach 222, 632
ForEach 622
foreach 문 99
Format 382
for 문 95
FQDN 140
Framework64 304
Freachable 큐 368
FreeCoTaskMem 365
from 632
Func 620
GAC 35
GAC_32 328
GAC_64 328
GAC_MSIL 328
gacutil.exe 326
GAC 폴더 326
GC.Collect 355, 359
GC.GetGeneration 355
GC.SuppressFinalize 369
get 146
GetAssemblies 552
GetBytes 393
GetConstructors 555
GetCustomAttributes 564
GetDirectoryName 426
GetEnumerator 221
GetEvents 555
GetExtension 426

GetFields 555
GetFileName 426
GetFileNameWithoutExtension 426
GetFullPath 426
GetHashCode 164, 181
GetInvalidFileNameChars 427
GetMethods 555
GetModules 553
GetPathRoot 426
GetProperties 555
getter 144
GetType 160
GetTypes 554
GetValue 166
GMT 375
goto 104
GotoExpression 629
group by 637
HasFlag 244
HasValue 599
HKEY_CLASSES_ROOT 569
HKEY_CURRENT_CONFIG 569
HKEY_CURRENT_USER 569
HKEY_LOCAL_MACHINE 569
HKEY_USERS 569
HTTP 통신 492
HTTP 헤더(Header)와 본문(Body) 494
ICollection〈T〉 589, 611
IComparable 409, 584, 636
IComparable.CompareTo 584
IComparable〈T〉 589
IComparer 219
IComparer〈T〉 589
IDisposable 361
IEnumerable 221, 592
IEnumerable〈T〉 589, 625, 632, 644
IEnumerator 221, 592
IEnumerator〈T〉 589
if 문 86
ILASM.EXE 27
implicit 190
in 242, 632
IndexExpression 629
IndexOf 379
Inherited 277
Initial Catalog 519
INSERT 510
int 59
Interlocked.Exchange 449
Interlocked.Increment 448
internal 143, 298
internal protected 143
InterNetwork 469
InterNetworkV6 469
int.Parse 345
IntPtr 571
InvocationExpression 629
IOException 418, 421, 422
IOrderedEnumerable〈TElement〉 644
IPAddress.Any 477
IPAddress.Loopback 480
IPAddress.Parse 465, 477
ipconfig 471
IPv4 464
IPv6 464
IP 주소 463
IronPython 657
is 156
IsBackground 433
IsMatch 391
IsNullOrEmpty 266, 373
join 638
Json 405
JustDecompile 52
KeyValuePair 605
Kind 376
KS_C_5601-1987 388

[L – R]

LabelExpression 629
LambdaExpression 629
Length 166
LINQ 630
LINQ to Objects 646
LINQ to SQL 646
LINQ to XML 646
LINQ 제공자 646
Listen 484
ListInitExpression 629
List〈T〉 576, 589
localhost 496

LocalMachine 569
lock 444
LOH 359
long 59
LoopExpression 629
Main 메서드 133
Marshal 585
MarshalAs 276
Marshal.SizeOf 585
MaxValue 250
MEF 567
MemberAssignment 629
MemberExpression 629
MemberInitExpression 629
MemberMemberBinding 629
Message 332
MessageBeep 288
MethodCallExpression 629
Microsoft.Win32 568
MinValue 250
Monitor.Enter 443
Monitor.Exit 443
mono–complete 43
MonoDevelop 44
mscorlib.dll 298
MSI 329
MSIL 27
nameof 695
namespace 137
nchar 508
netmodule 32
NetworkStream 669
new 178
NewArrayExpression 630
NewExpression 630
new 연산자 113
NHibernate 547
NonSerialized 402
Non–Signal 451
null 72
nullable 598
null 조건 연산자 689
nvarchar 508
nvarchar(max) 509
object 159
Obsolete 310
OpenSubKey 569
orderby 636
ORM 547
OR 매핑 547
out 239
OUT 242
OutOfMemoryException 365
override 177
ParameterExpression 628
params 286
Parse 346
partial 597
partial 클래스 598
Path.Combine 426
Path.GetInvalidPathChars 427
Path.GetRandomFileName 427
Path.GetTempFileName 427
Path.GetTempPath 427
PDB 313
ping 471
PInvoke 289
POCO 529
Pop 413
Position 396
POSIX time 377
Predicate 델리게이트 624
private 142
PrivateMemorySize64 365
Process.GetCurrentProcess 365
protected 142, 151
protected internal 143
protected set 607
ProtocolType 472
public 115, 143, 298
PublicKeyToken 323
Push 413
PUSH 347
Queue〈T〉 589
Rank 166
ReadAllBytes 421
ReadAllLines 421
ReadAsync 666, 670
readonly 247
real 509
Receive 485
ReceiveFrom 478

ref 234
Registry 568
RegistryKey 568
Registry.LocalMachine 569
Replace 379, 391
return 116, 119
RISC 프로세서 394

[S – Z]

sbyte 58
sealed 152
SearchOption.AllDirectories 425
select 632
Select 626
SELECT 513
Send 485
SendTo 478
Serializable 401
Serialize 402
set 146
setter 144
short 58
Signal 451
sln 294
smallint 509
sn.exe 322
SocketType 472
Sort 166
SortedDictionary<TKey 589
Source 332
Split 379
Spring.NET 567
SqlCommand.CommandText 521
SQLExpress 502
SQL Server Management Studio 504
sqlservr.exe 504
SqlTransaction 549
SqlTransaction.Abort 549
SQL 주입 526
SQL 쿼리 510
SSMS 504
stackalloc 292
StackFrame 696
StackOverflowException 351
Stack<T> 589
StackTrace 332, 337
StartsWith 379
static 131, 602
static constructor 135
Stream 395
StreamWriter 418
string 60
StringComparision.OrdinalIgnoreCase 382
StringComparison 382
string.ToLower 386
SUBST 328
Substring 379
supportedRuntime 306
SwitchCase 630
SwitchExpression 630
switch 문 89
syntactic sugar 148
System.ApplicationException 331, 339
System.Array 165
System.Attribute 272
System.AttributeUsageAttribute 274
System.BitConverter 393
System.Boolean 64
System.Byte 58
System.Char 60
System.Collections 407
System.Collections.ArrayList 407
System.Collections.Generic 588
System.Collections.Hashtable 410
System.Collections.Queue 415
System.Collections.SortedList 413
System.Collections.Stack 413
System.currentTimeMillis 377
System.Data.DataColumn 534
System.Data.DataSet 534, 535
System.Data.DataTable 534
System.Data.IDataReader 517
System.Data.IDbCommand 517
System.Data.IDbConnection 517
System.Data.IDbDataAdapter 517
System.Data.IDbDataParameter 517
System.Data.Linq 646
System.Data.SqlClient 518
System.Data.SqlClient.SqlCommand 520
System.Data.SqlClient.SqlConnection 518
System.Data.SqlClient.SqlDataAdapter 528

System.Data.SqlClient.SqlDataReader 523
System.Data.SqlClient.SqlParameter 525
System.DateTime 374
System.Decimal 59
System.Delegate 200, 459
System.Diagnostics.Stopwatch 378
System.DivideByZeroException 333
System.Double 59
System.EventArgs 254
System.Exception 331
System.IndexOutOfRangeException 311, 331, 590
System.Int16 58
System.Int32 59
System.Int64 59
System.IO.BinaryReader 399
System.IO.BinaryWriter 399
System.IO.Directory 423
System.IO.DirectoryInfo 423
System.IO.File 421
System.IO.FileInfo 421
System.IO.FileStream 416
System.IO.MemoryStream 395
System.IO.Path 426
System.IO.StreamReader 398
System.IO.StreamWriter 398
System.Linq 616
System.Linq.Expressions.Expression 627
System.MulticastDelegate 200
System.Net.Dns 468
System.Net.HttpWebRequest 497
System.Net.IPAddress 464
System.Net.IPEndPoint 468
System.Net.Sockets.Socket 472
System.Net.WebClient 497
System.Nullable〈T〉 598
System.NullReferenceException 330
System.Numerics 571
System.Object 159
System.OverflowException 285
System.Runtime.InteropServices 585
System.Runtime.Serialization.dll 406
System.Runtime.Serialization.Formatters.Binary.BinaryFormatter 401
System.Runtime.Serialization.Json.DataContractJsonSerializer 405
System.Sbyte 58
System.Security.SecurityException 570
System.Single 59
System.String 60, 378
System.SystemException 331
System.Text.Encoding 388
System.Text.RegularExpressions.Regex 389
System.Text.StringBuilder 386
System.Threading.EventWaitHandle 451
System.Threading.Interlocked 448
System.Threading.Monitor 439
System.Threading.Thread 431
System.Threading.ThreadPool 449
System.TimeSpan 377
System.Transactions.dll 550
System.Transactions.TransactionScope 550
System.UInt16 59
System.UInt32 59
System.UInt64 59
System.ValueType 158
System.Xml.Linq 646
System.Xml.Serialization.XmlSerializer 403
Systme.FormatException 345
Task 671
TaskFactory 672
Task.Factory.StartNew 673
Task〈TResult〉 671
Task.WaitAll 678
TcpClient 669
TCP 소켓 472, 483
this 168, 615
Thread 600
Thread.CurrentThread 431
Thread.CurrentThread.ManagedThreadId 666
Thread.Join 434
ThreadPool.QueueUserWorkItem 450
Thread.Sleep 431
Thread.Start 434
ThreadState 431
throw 339
throw ex 340
Ticks 374
tinyint 509
ToArray 397
ToLower 379
ToString 159, 181, 332
TotalDays 378

TotalHours 378
TotalMilliseconds 378
TotalMinutes 378
TotalSeconds 378
ToUpper 379
Trace 316
TRACE 315
try 332
TryExpression 630
TryParse 241, 346
TValue〉 589
TypeBinaryExpression 630
Typed DataSet 540
Type.EmptyTypes 559
Type.GetMembers 554
Type.GetType 559
Type.Name 565
typeof 161
UDP 서버 479
UDP 소켓 472, 475
UDP 클라이언트 481
uint 59
ulong 59
UnaryExpression 630
unchecked 284
Unicode 389
Unity 컨테이너 567
Unix Time 377
Unix Timestamp 377
unsafe 289
UPDATE 516
UploadFile 498
URL 492
Users 569
ushort 59
using 139, 363, 616
using static 685
UTC 375
UTF–7 388
UTF–8 388
UTF8 389
UTF–16 388
UTF–32 388
UTF32 389
v2.0.50727 304
v3.0 304
v3.5 304
v4.0.30319 304
value 147
var 604, 653
varchar 508
varchar(max) 508
virtual 177
Visual Studio Community 47
void 116
VS2015용 개발자 명령 프롬프트 322
WaitOne 451
when 699
where 583, 634
Where 625
where T: class 585
where T: new( ) 585
where T: struct 585
where T: U 585
while 문 100
Win32 API 730
Windows Forms Application 300
WPF Application 300
WriteAllBytes 421
WriteAllLines 421
WriteAsync 670
WriteXml 528
www.pinvoke.net 289
x64 319
x86 319
XAttribute 646
XCopy 배포 325
XDocument 646
XElement 646
XML 294
yield break 592
yield return 592, 632
zero–based index 78