Hello World!

2023. Spring

국민대학교 소프트웨어학부

ASCII 문자

NUL	null

HT	Horizontal Tab	가로 탭
LF	Line Feed	줄 바꿈
VT	Vertical Tab	세로 탭
FF	Form Feed	폼 피드(새 페이지로 이동)
CR	Carriage Return	캐리지 리턴(첫 칸으로 이동)

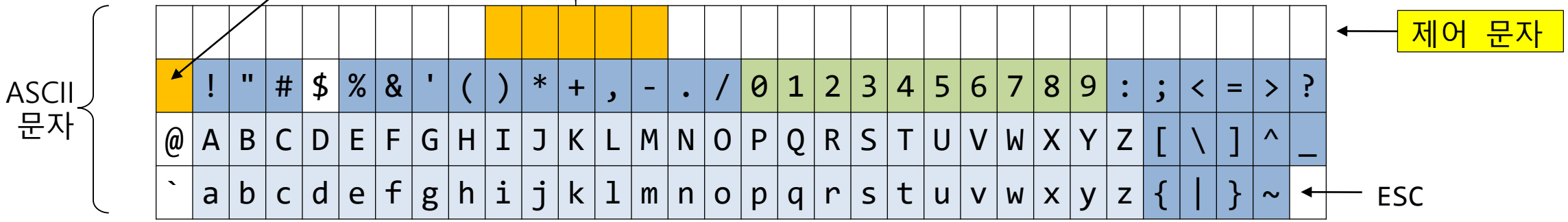
10진수	ASCII	10진수	ASCII	10진수	ASCII	10진수	ASCII
0	NUL	32	Space	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107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

제어문자

기본 소스 문자 집합

- 총 97문자
  - 영문자 대/소문자 (52)
  - 숫자 (10)
  - 특수문자 (29) : _ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " '
  - 공백(white space)문자(6) :

스페이스, 가로 탭, 줄 바꿈, 세로 탭, 폼 피드, 캐리지 리턴



주석 (Comment)

- 주석
  - 프로그램의 일부이지만 실행되지 않는 영역
  - 프로그램을 이해할 수 있도록 설명해 놓은 부분
  - 컴파일러는 주석을 스페이스 문자 한 개로 처리함

```
1   /*********************************************
2    * HelloWorld.cpp: Hello World를 출력하는 프로그램
3    *********************************************/
4   #include <iostream>
5
6   using namespace std;
7
8   int main()
9   {                                       // 함수 시작
10      cout << "Hello World!\n";
11
12      return 0;
13  }                                       // 함수 종료
```

```
1   // line splicing 의 예 \
2   (MACRO 정의)
3   #define PRINT_SUM (X, Y)        \
4       do {                        \
5           cout << ((X) + (Y));    \
6       } while(0)
```

  - 주석 방법 (1)
    - /* 로 시작, */ 로 종료
    - 중첩할 수 없음
  - 주석 방법 (2)
    - // 로 시작
    - 백 슬래시(\)가 바로 앞에 오지 않는 새 줄 문자가 올 때 주석을 종료함
      - 백 슬래시 다음에 새 줄 문자가 오는 경우는 다음 줄을 연속해서 한 줄로 연결하는 역할을 함(line splicing)

식별자 (Identifier)

- 식별자(identifier)
  - 변수 이름, 함수 이름, 클래스 이름 등의 이름을 말함

- 식별자를 만드는 규칙
  - 영문자 대/소문자, 숫자, 밑줄(_) 을 조합하여 만듦
    - 영문자 대/소문자는 구별함
  - 숫자로 시작해서는 안됨
  - 키워드(keyword)는 식별자로 사용할 수 없음
  - 예약된 식별자(reserved identifiers)
    - 다음과 같은 이름은 예약되어 있음 (일반 사용자는 사용하지 말 것)
      - 밑줄 한 개로 시작하고 그 다음에 영문자 대문자로 시작하는 이름 (예: `_Reserved`)
      - 연속된 두 개의 밑줄을 포함하는 경우 (예: `__reserved`, `reserved__identifier`)
    - 다음의 경우도 다른 용도로 예약되어 있음 (global namespace에서 예약되어 있음)
      - 밑줄 한 개로 시작하는 이름

식별자 (Identifier)

사용 가능한 식별자의 예	
alphanum AlphaNum alpha_num Alpha_Num Alpha26 alpha_26_num_10_	alphanum 과 다른 식별자임

사용 불가능한 식별자의 예	
alpha num	space가 있으면 안됨
26Alphabet	숫자로 시작하면 안됨
alpha&num	특수문자 &를 사용하면 안됨
%percent	특수문자 %를 사용하면 안됨
while	키워드는 사용하지 못함

예약된 식별자 (Reserved Identifier)

```
1  #ifndef _MY_HEADER_H_    // __MY_HEADER_H__
2  #define _MY_HEADER_H_
3
4  ...
5  // my_header.h 내용
6  ...
7  #endif // _MY_HEADER_H_
```

```
1  #ifndef MY_HEADER_H_    // MY_HEADER_H__
2  #define MY_HEADER_H_
3
4  ...
5  // my_header.h 내용
6  ...
7  #endif // MY_HEADER_H_
```

```
1  #include <cstddef>
2
3  static const std::size_t _max_size = 256;
4
5  unsigned int get_size(unsigned int size) {
6      return size < _max_size ? size : _max_size;
7  }
```

```
1  #include <cstddef>
2
3  static const std::size_t max_size = 256;
4
5  unsigned int get_size(unsigned int size) {
6      return size < max_size ? size : max_size;
7  }
```

키워드(Keyword)

- 키워드(keyword)
  - 특별한 의미가 부여되어 사전에 미리 예약된 식별자
  - 식별자로 사용할 수 없음

C로 부터 사용된 키워드		
auto	extern	sizeof
break	float	static
case	for	struct
char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	volatile
else	short	while
enum	signed	

32

C++ 추가 키워드		
asm	inline	this
bool	mutable	throw
catch	namespace	true
class	new	try
const_cast	operator	typeid
delete	private	typename
dynamic_cast	protected	using
explicit	public	virtual
export	reinterpret_cast	wchar_t
false	static_cast	
friend	template	

31

대체연산자	
and	&&
and_eq	&=
bitand	&
bitor	|
compl	~
not	!
not_eq	!=
or	||
or_eq	|=
xor	^
xor_eq	^=

11

C++11
alignas
alignof
char16_t
char32_t
constexpr
decltype
noexcept
nullptr
static_assert
thread_local

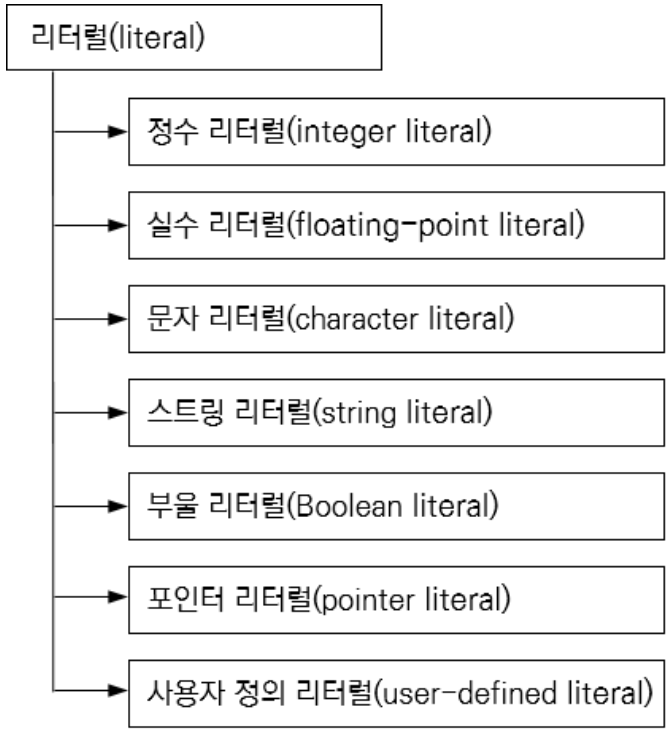
10

문장 부호

- 문장부호(28개)
  - ! % ^ & * ( ) - + = { } | ~ [ ] \ ; ' : " < > ? , . / #
  - 특수 문자 29개 중에서 식별자를 만드는데 사용되는 밑줄(_) 제외한 모든 문자

리터럴(Literal)

- 리터럴(literal)
  - 프로그램 내에서 그 자체로 고정된 값을 표현하는 데이터
  - 예: `100`, `3.14`, `'a'`, `"Hello World"`, `true`, `nullptr`, `42.195_km`
  - 리터럴의 7종류



정수 리터럴(Integer Literal)

- 정수 리터럴(integer literal)
  - 숫자로 만들어진 정수값
  - 접두어(prefix)와 접미어(postfix)를 붙여서 표현됨
    - 접두어 : 정수 리터럴을 표현하는 진수(base)를 나타내는데 사용
    - 접미어 : 정수 리터럴을 저장하는 자료형을 나타내는데 사용

정수 리터럴(Integer Literal)

- 정수 리터럴의 접두어
  - 정수 리터럴을 표현하는 진수(base)를 나타내는데 사용
    - 10진(decimal) 정수 리터럴
      - 10진 숫자(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)로 표현
      - 단 가장 높은 자리수는 0이 아닌 10진 숫자여야 함
      - 특별한 접두어는 붙이지 않음
    - 2진(binary) 정수 리터럴
      - 이 정수들은 2진 숫자(0, 1)로 표현
      - 반드시 접두어 0b 혹은 0B로 시작
    - 8진(octal) 정수 리터럴
      - 8진 숫자(0, 1, 2, 3, 4, 5, 6, 7)로 표현
      - 반드시 접두어 0으로 시작하여야 함
    - 16진(hexadecimal) 정수 리터럴
      - 16진 숫자(0~9, a~f, A~F)로 표현
      - 반드시 접두어 0x 혹은 0X로 시작하여야 함
      - 16진 숫자 a/A, b/B, c/C, d/D/, e/E, f/F 는 각각 10, 11, 12, 13, 14, 15를 나타냄

접두어	진수(base)	예
없음	10진수(decimal)	256, 1024, 9999
0b, 0B	2진수(binary)	0b11111111, 0B1010
0	8진수(octal)	0123, 07777, 0256
0x, 0X	16진수(hexadecimal)	0xFFFF, 0X0001, 0x90AbcD

정수 리터럴(Integer Literal)

- 정수 리터럴의 접미어
  - 정수 리터럴을 저장하는 자료형을 나타내는데 사용
  - 접미어에 사용되는 각 문자들 u/U, l/L은 대문자와 소문자를 구별하지 않고 동일하게 사용됨

접미어	자료형	예
없음	int long int long long int	12345, 2147483647 2147483648
u, U	unsigned int unsigned long int unsigned long long int	4294967295u 4294967296u
l, L	long int unsigned long int long long int	2147483647L 4294967295L 4294967296L
ul, UL	unsigned long int unsigned long long int	4294967295UL 4294967296UL
ll, LL	long long int	9223372036854775807LL
ull, ULL	unsigned long long int	18446744073709551615ULL

- 주어진 정수 리터럴을 저장하는 자료형은 그 정수를 저장하는 가장 작은 메모리를 차지하는 자료형으로 정의됨
- 예를 들어 2147483647(=2^{31}-1) 은 4-바이트(32-비트)로 표시될 수 있으므로 int 자료형으로 지정됨
- 이 정수 보다 1만큼 큰 정수 2147483648(=2^{32})은 int 자료형으로는 저장할 수 없으므로 int 보다 큰 8-바이트(64-비트) 메모리를 차지하는 long long int 자료형으로 정의됨

실수 리터럴(Floating-point Literal)

- 실수 리터럴(floating-point literal)
  - 실수값을 표현하는데 사용
  - 실수 리터럴을 만드는 규칙
    - 실수 리터럴은 정수 부분, 소수점, 소수 부분, 지수 부분으로 구성됨
    - 지수 부분은 문자 e혹은 E로 시작하여 부호(+,-)가 앞에 붙을 수 있는 정수로 만들어 짐
      (지수 부분은 10의 거듭제곱을 나타냄)
    - 정수 부분과 소수 부분은 각각 연속된 10진 숫자로만 구성되고 두 부분 중에서 한 부분은 생략될 수 있음
    - 소수점 혹은 지수 부분 중의 하나는 생략될 수 있음
    - 마지막에 자료형을 나타내는 접미사 문자 f/F, l/L중의 하나가 붙을 수 있음

```
31.4159
31.             /* 소수 부분이 생략됨 */
.314159         /* 정수 부분이 생략됨 */
0.314159e2      /* 31.4159 */
3.14159E+1      /* 31.4159 */
3141.59e-2      /* 31.4159 */
314159E-5       /* 31.4159, 지수 부분이 있어서 소수점이 생략될 수 있음 */
.00314159e+4    /* 31.1459, 정수 부분이 생략됨 */
```

실수 리터럴(Floating-point Literal)

- 실수 리터럴(floating-point literal)
  - 실수 리터럴은 기본적으로 `double` 자료형임 (주의: `float` 자료형 아님)
    - 접미사 `f` (혹은 `F`)를 붙여서 `float` 형으로 만들 수 있음
    - 접미사 `l` (혹은 `L`)을 붙여서 `long double`형으로도 만들 수 있음
  - 실수 리터럴은 제일 앞에 붙을 수 있는 부호(+,-)를 포함하지 않음
    - 부호는 별도로 단항 연산자로 처리

문자 리터럴(Character Literal)

- 문자 리터럴(character literal)
  - 한 개의 문자를 표현하는데 사용
  - 작은 따옴표('')로 묶어 만듦
    - 예: `'A', 'a', '0', '_', '$'`

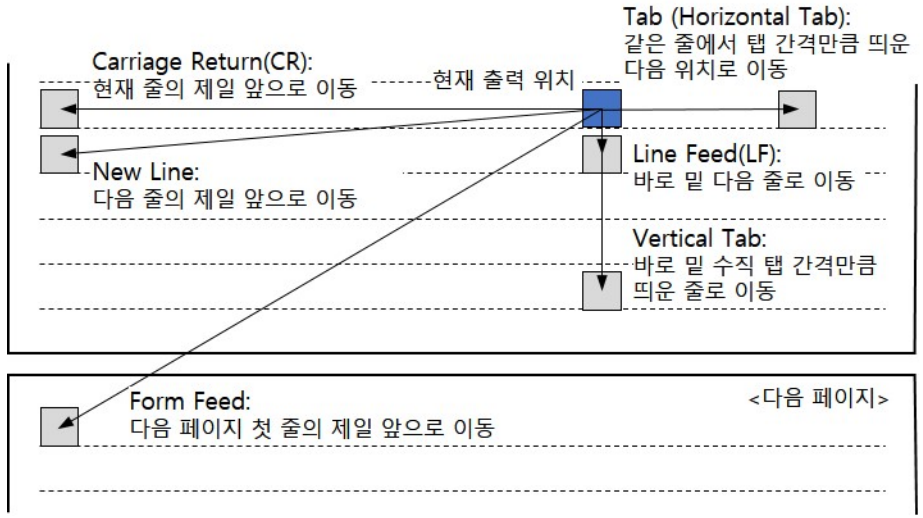
문자 리터럴(Character Literal)

- 이스케이프 시퀀스(escape sequence)
  - 나타낼 심볼이 없어서 문자 상수로 표시하기 어려운 문자를 표시하기 위하여 사용함
    - 예: 제어문자 중에서 줄 바꿈(new line), 탭, 캐리지 리턴 등
  - 백 슬래시 문자(\) 바로 뒤에 한 개의 문자로 만들어지거나 한 개 이상의 숫자들로 만들어진 문자 조합

이스케이프 시퀀스	문자	이름		설명
\n	NL(LF)	줄 바꿈	new line	다음 줄의 처음으로 이동함
\t	HT	탭	horizontal tab	같은 줄에서 다음 탭 스톱으로 이동함
\v	VT	수직 탭	vertical tab	밑으로 수직 탭 스톱 줄로 이동함
\b	BS	백 스페이스	backspace	같은 줄에서 뒤로 한 칸 이동함
\r	CR	캐리지 리턴	carriage return	같은 줄의 처음으로 이동함
\f	FF	페이지 넘김	form feed	다음 페이지의 제일 첫 부분으로 이동함
\a	BEL	경보음	audible alert	경보음
\\	\	백슬래시	backslash	백슬래시
\?	?	물음표	question mark	물음표
\'	'	작은 따옴표	single quote	작은 따옴포
\"	"	큰 따옴표	double quote	큰 따옴표
\ooo	*ooo*	정수(8진수)	octal number	8진수 정수 (*o*는 8진수 숫자를 표시함)
\x*hh*	*hh*	정수(16진수)	hex number	16진수 정수 (*h*는 16진수 숫자를 표시함)

문자 리터럴(Character Literal)

- 이스케이프 시퀀스 `\n \t \v \r \f`



원래의 의미상으로
New Line(NL) = Carriage Return(CR) + Line Feed(LF)

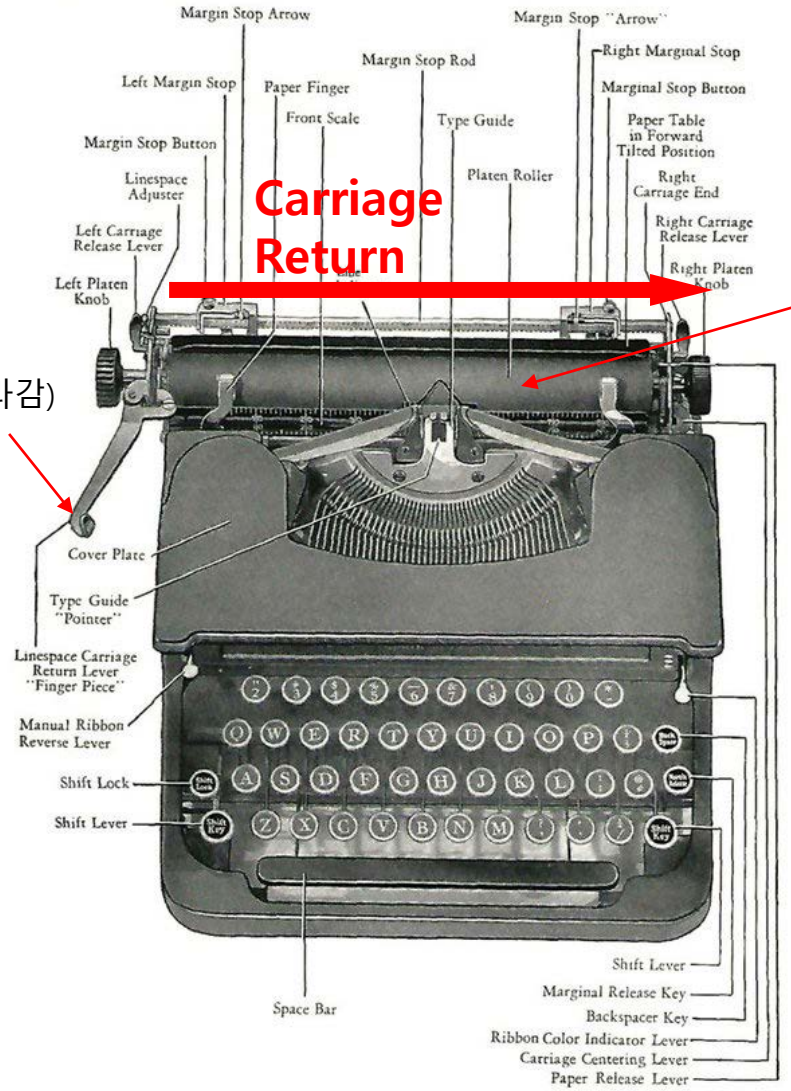
LF의 원래 의미는 없어지고 NL에 통합됨

UNIX, Linux, Mac OS 계열
\n = LF

윈도우즈 계열
\n = CR + LF

문자 리터럴(Character Literal)

- Carriage Return과 Line Feed
  - 타자기로부터 유래함



Line Feed
(옆으로 누르면
종이가 한 줄 올라감)

Carriage

https://www.quora.com/What-is-a-carriage-on-a-typewriter

문자 리터럴(Character Literal)

- Carriage Return과 Line Feed

https://www.youtube.com/watch?v=FkUXn5bOwzk

Typewriter
2:50 Carriage Return
3:35 Line Feed (옆으로 누르면 종이가 한 줄 올라감)
3:10 Tab Stop Set

https://www.youtube.com/watch?v=2XLZ4Z8LpEE

Teletype (Console)
10:00 Carriage Return + Line Feed

문자 리터럴(Character Literal)

- Carriage Return과 Line Feed

문자 리터럴(Character Literal)

- Carriage Return과 Line Feed

UNIX, Linux, Mac OS 계열 \n = LF	윈도우즈 계열 \n = CR + LF

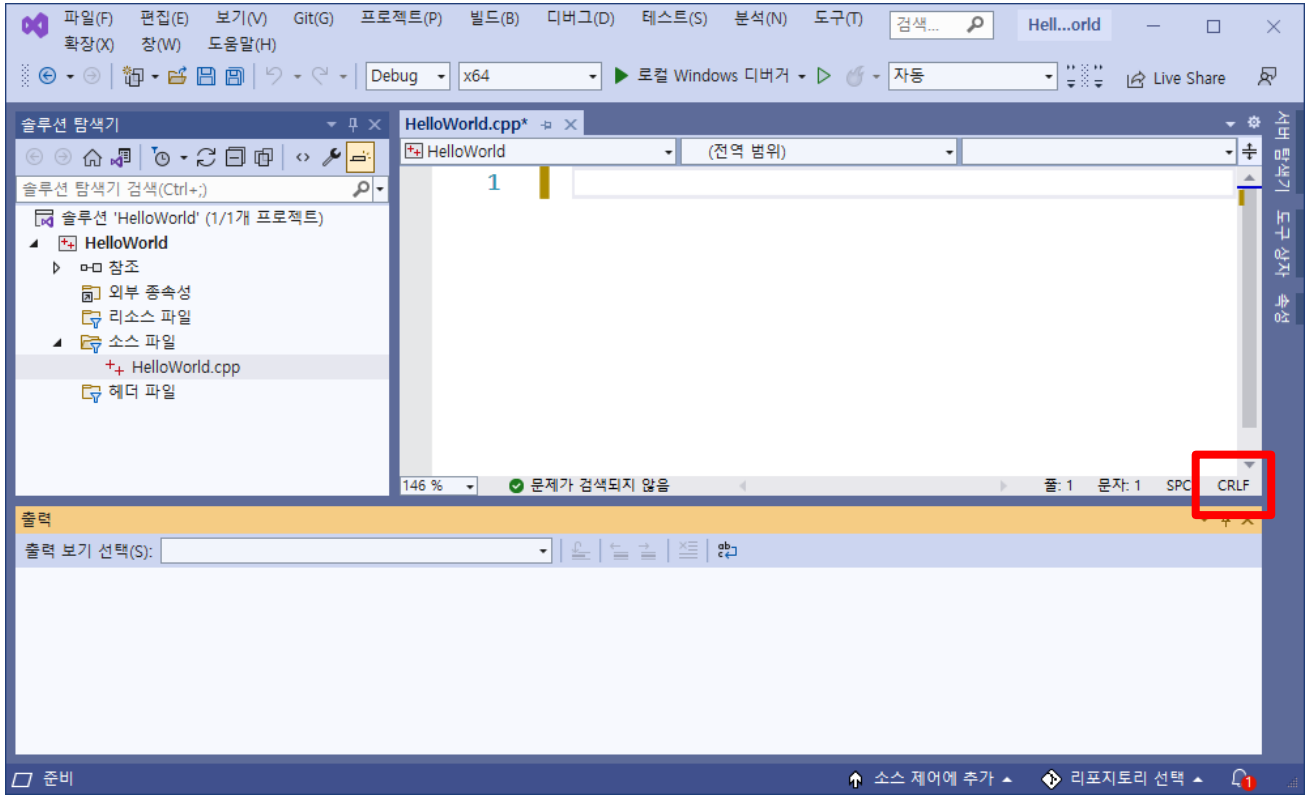
```
a ↲
1 ↲
# ↲
```

UNIX, Linux, Mac OS 계열

a	LF	1	LF	#	LF

윈도우즈 계열

a	CR	LF	1	CR	LF	#	CR	LF



문자 리터럴(Character Literal)

- 이스케이프 시퀀스 \\ \' \"
  - 백 슬래시 문자(\)를 스트링에 삽입하는 경우

```
"backslash \and other chars"
```

```
"backslash \\and other chars"
```

  - 작은 따옴표를 문자 리터럴로 표시하는 경우

```
'''
```

```
'\''
```

  - 큰 따옴표를 스트링 리터럴에 표시하는 경우

```
"He said to me, "How are you?""
```

```
"He said to me, \"How are you?\""
```

문자 리터럴(Character Literal)

- 이스케이프 시퀀스 `\ooo`
  - 백슬래시 다음에 숫자가 1, 2, 3개로 만들어진 8진수로 표현
  - 어떤 문자를 8진수의 코드값으로 나타낸 것
  - 예: `'\101'` (영문자 `'A'`), `'\007'` `'\07'` `'\7'` (경보음 문자 `'\a'`)

- 이스케이프 시퀀스 `\xhh`
  - 백슬래시 다음에 문자 x (문자 X는 아님) 그 다음에 16진수로 표현
  - 어떤 문자를 16진수의 코드값으로 나타낸 것
  - 예: `'\x41'` (영문자 `'A'`), `'\x07'` `'\x7'` (경보음 문자 `'\a'`)

스트링 리터럴(String Literal)

- 스트링 리터럴(string literal)
  - 큰 따옴표("")로 둘러싸인 연속된 문자들
    - 예: "Hello World!"
  - 엔터키를 눌러 여러 줄로 표현할 수 없음
    - 줄 바꿈하기 전에 백 슬래시 문자(\)를 두면 마치 한 줄에 입력한 것처럼 처리함

```
"A long string literal
can be broken up
over multiple lines"
```

```
"A long string literal \
can be broken up \
over multiple lines"
```

```
"A long string literal can be broken up over multiple lines"
```

```
cout << "A long string literal \ncan be broken up \nover multiple lines";
```

```
A long string literal
can be broken up
over multiple lines
```

스트링 리터럴(String Literal)

- 긴 스트링 리터럴
  - 여러 개의 스트링으로 나누고, 여러 줄에 배치할 수 있음
  - 전처리기가 연속된 여려 개의 스트링을 이어 붙여서 한 개의 스트링으로 만듦

```
cout << "A long string literal "
        "can be broken up "
        "over multiple lines";
```

```
cout << "A long string literal can be broken up over multiple lines";
```

부울 리터럴(Boolean Literal)

- 부울 리터럴(Boolean literal)
  - 두 개의 리터럴만 존재함
    - `true, false` (키워드로 표시됨)

포인터 리터럴(Pointer Literal)

- 포인터 리터럴(pointer literal)
  - 단 한 개의 리터럴만 존재함
    - `nullptr` (키워드로 표시됨)
    - `nullptr`은 널 포인터(null pointer) 상수를 나타냄

사용자 정의 리터럴(User-defined Literal)

- 사용자 정의 리터럴(user-defined literal)
  - 앞에서 설명한 정수, 실수, 문자, 스트링 리터럴에 사용자가 지정한 접미어(suffix)를 붙여서 프로그래머가 정의하는 리터럴을 만들 수 있게 함
  - 예 :
    - 거리를 나타내는 실수 리터럴에 문자 언더스코어 '_'와 단위를 나타내는 접미어를 붙여서 42.195_km, 1000.0_m 등과 같이 만들 수 있게 해 줌
    - 사용자 정의 리터럴에 적용할 수 있는 연산자를 정의하면 단위가 다른 실수 리터럴을 단위에 맞게 계산할 수 있도록 해 줌
    - 예: 42.195_km + 1000.0_m 는 43195로 계산됨.

공백(White Space) 문자

- 공백(white space) 문자
  - 공백
    - 인쇄된 문서에서 단어와 단어 사이 혹은 줄 사이의 띄어쓰기에 사용됨
    - 문서를 읽기 쉽게 하는 역할을 수행함
  - C++ 공백문자 (6개)
    - 공백 문자는 서로 인접한 토큰을 구분하는 역할도 수행함
    - 컴파일러는 이들 공백 문자를 인접한 토큰을 구분하기 위해서만 사용하고 공백 문자 그 차체는 무시하고 사용하지 않음

Char	문자	이름		설명
' '		스페이스	space	빈 칸
'\n'	NL(LF)	줄 바꿈	new line	다음 줄의 처음으로 이동함
'\t'	HT	탭	horizontal tab	같은 줄에서 다음 탭 스톱으로 이동함
'\v'	VT	수직 탭	vertical tab	밑으로 수직 탭 스톱 줄로 이동함
'\r'	CR	캐리지 리턴	carriage return	같은 줄의 처음으로 이동함
'\f'	FF	페이지 넘김	form feed	다음 페이지의 제일 첫 부분으로 이동함

    - 주석은 컴파일러에 의해서 한 개의 스페이스로 변환되므로 공백 문자로 취급함

토큰(Token)

- 토큰(token)
  - C++ 프로그램에서 문법적으로 의미가 있으며 더 이상 나눌 수 없는 최소의 기본 요소
  - C++ 프로그램은 토큰과 공백문자로 구성되어 있음
  - 토큰 (크게 5종류)
    - 키워드
    - 식별자
    - 리터럴(7가지)
    - 연산자
    - 구분문자(punctuators) 혹은 문장부호(28개)

      ! % ^ & * ( ) - + = { } | ~ [ ] \ ; ' : " < > ? , . / #

토큰(Token)

- 예

```
int main()
{
    int a, b, c;

    a = 1;
    b = 2;
    c = a + b;
    printf("%d + %d = %d", a, b, c);
    return 0;
}
```

키워드	int return
식별자	main a b c printf
리터럴	1 2 "%d + %d = %d" 0
연산자	= + ()
구분문자	{ } , ;

소스로 부터 실행파일로 번역단계

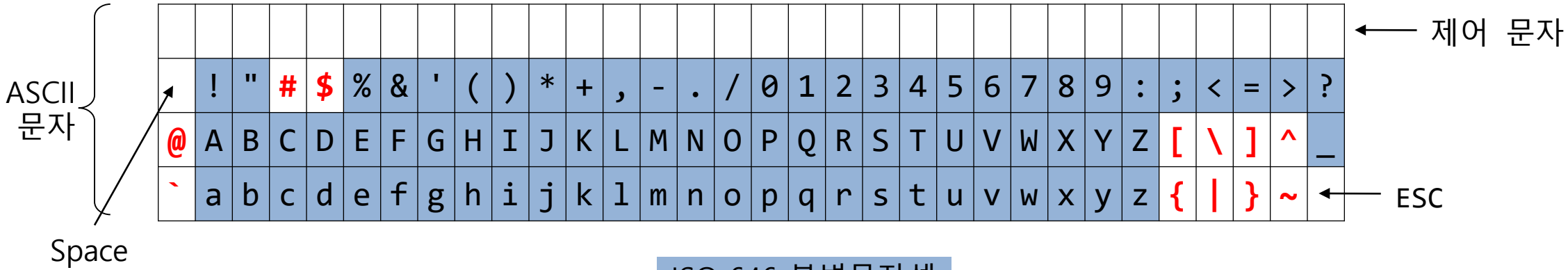
- Character Mapping
  - Trigraph(삼중자) 문자를 1개의 문자로 변환 (삼중자는 C++17부터 제외)
    - `??=, ??/, ??', ??(, ??), ??!, ??<, ??>, ??-`
- Line Splicing
  - 백 슬래시 문자(\)로 끝나는 줄(바로 뒤에 newline 문자가 오는 줄)을 내부적으로 '\'과 newline 없이 다음 줄과 연결하여 한 줄로 만듦.
- Tokenization(토큰화)
  - 소스파일을 전처리용 토큰과 공백문자로 분할함.
  - 주석(comment)는 한 개의 스페이스 문자로 대치함
  - newline 문자는 그대로 둠

소스로 부터 실행파일로 번역단계

- Preprocessing
  - 전처리기 지시문을 모두 처리함
  - 마크로는 소스파일에 확장해 둠
- Character-set Mapping
  - 모든 문자와 이스케이프 시퀀스 문자를 ASCII 문자로 변환함
    - 예: `\n, \r, \t, \f` 등
- String Concatenation
  - 연속된 스트링은 이어 붙여서 하나의 긴 스트링으로 만듦
    - 예: `"String " "Concatenation"` → `"String Concatenation"`
- Compilation
  - 소스 코드를 오브젝트 코드로 번역
- Linkage
  - 최종적으로 실행파일을 만듦

ASCII 문자 및 ISO 646 문자

- ASCII & ISO 646
  - ISO 646 불변문자셋(invariant character set)
    - ISO 646: 7-bit 문자 코드 셋을 정의함 (1960년대에 제정됨)
    - 불변문자셋: 라틴계열 언어를 사용하는 모든 국가에서 공통으로 사용할 문자 셋
      - 82개 문자로 구성 (제어문자, 스페이스, ESC 제외)
      - 나머지 문자코드는 각 국가별로 고유한 문자를 표시하기 위해 사용함
        - » 영국 : # → £
        - » 덴마크 : [ ] { } | \ → Æ Å æ å ø Ø



ISO 646 불변문자셋

Trigraph (삼중자)

- Trigraph
  - C++(C 포함) 언어의 소스 코드는 ISO 646 불변문자셋에 포함되 않은 9개의 문자를 사용함.
    - # \ ^ [ ] | { } ~
  - 또한 초기의 키보드에는 키가 없는 ASCII 코드 문자도 존재함
    - ^ ~
  - 이러한 문자를 표시하기 위한 방법

Trigraph (삼중자)

- Trigraph
  - 세 개의 문자를 조합하여 키보드에 없는 한 문자로 치환하여 사용함
    - 세 개의 문자 조합을 trigraph(삼중자) 라고 부름
    - trigraph는 다음과 같이 첫 두 문자는 ?? 임.
  - C++17부터는 표준에서 제외됨
    - Visual Studio 2022에서도 default로 trigraph를 사용하지 않음 (옵션: /Zc:trigraphs 사용해야 함)

삼중자	치환 문자
??=	#
??/	\
??'	^
??(	[
??)	]
??!	|
??<	{
??>	}
??-	~

소스코드를 실행파일로 빌드하는 과정에서 전처리 이전에 제일 먼저하는 작업이 삼중자를 치환문자로 바꾸는 것임.

```
??=include <iostream>
using namespace std;

int main()
??<
    cout << "Hello World!??/n";
    return 0;
??>
```

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello World!\n";
    return 0;
}
```

Trigraph (삼중자) 사용의 문제점

- 문제점의 예
  - 주석문과 line splicing 이 일어날 수 있음

```
#include <iostream>
using namespace std;

int main()
{   // 어떤 메시지를 출력해야 하나요????/
    cout << "Hello World!\n";
    return 0;
}
```

```
#include <iostream>
using namespace std;

int main()
{   // 어떤 메시지를 출력해야 하나요??\
    cout << "Hello World!\n";
    return 0;
}
```

- line splicing: 백 슬래시 문자(\)로 끝나는 줄(바로 뒤에 newline 문자가 오는 줄)을 내부적으로 '\'과 newline 없이 다음 줄과 연결하여 한 줄로 만듦.
- 따라서 위의 예에서는 주석문의 다음 줄이 주석문과 한 줄로 합쳐져서 주석문으로 처리되어 실행되지 않음

Trigraph (삼중자) 사용의 문제점

- 문제점의 예
  - 주석문과 line splicing 이 일어날 수 있음

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello World!\n";
    return 0;
}
```

  - 백 슬래시 문자(\)로 끝나는 줄(바로 뒤에 newline 문자가 오는 줄)을 내부적으로 '\'과 newline 없이 다음 줄과 연결하여 한 줄로 만듦.
  - Because trigraphs are processed early, a comment such as // Will the next line be executed?????/ will effectively comment out the following line, and the string literal such as "Enter date ??/??/??" is parsed as "Enter date ₩₩??".
  - https://en.cppreference.com/w/cpp/language/operator_alternative

Trigraph (삼중자) 사용의 문제점

- 문제점의 예
  - 다음과 같은 코드는 의도하지 않게 출력됨

```
...
    cout << "Enter date ??/??/??"
...
```

```
...
    cout << "Enter date \\??"
...
```

  - 출력

```
Enter date \??
```

이스케이프 시퀀스 \?

- 왜 이스케이프 시퀀스 \? 가 필요한가?
  - 삼중자로 사용되는 것을 방지하기 위함.
  - 예

```
...
    cout << "Enter date ?\?/?\?/??"
...
```

```
Enter date ??/??/??
```

  - 삼중자로 사용될 수 없는 경우에는 ? 를 그대로 사용하면 됨.
    - 이 경우에는 이스케이프 시퀀스 \? 를 사용할 필요 없음

```
cout << "어떤 메시지를 출력해야 하나요????"
```

대체 연산자 / 이중자

- 대체 연산자 (alternative operators), 이중자(digraph)
  - ISO 646 불변문자셋에 포함되지 않는 문자를 사용하는 연산자 기호나 구분자와 같은 특수 기호를 대체하기 위해 만듦

대체연산자	연산자
and	&&
and_eq	&=
bitand	&
bitor	|
compl	~
not	!
not_eq	!=
or	||
or_eq	!=
xor	^
xor_eq	^=

이중자	치환 토큰
<%	{
%>	}
<:	[
:>	]
%:	#
%:%:	##

```
%:include <iostream>
using namespace std;

int main()
<%
    cout << "Hello World!\n";
    return 0;
%>
```

- 문자 & 와 ! 는 불변문자셋에 포함되지 않지만 ISO 646보다 더 엄격한 문자셋을 정의한 경우도 수용하기 위함
- 대체 연산자는 키워드에 포함됨