제 4 장
변수와 자료형 3부

오류탐지 및 수정 코드
(Error Detection & Correction)

오류 탐지 및 수정 코드

- **Parity bit**
  - even parity, odd parity
  - **7비트 ASCII 코드에 패리티 비트를 추가한 코드**

데이터	짝수패리티	홀수패리티
…	…	…
A	0 1000001	1 1000001
B	0 1000010	1 1000010
C	1 1000011	0 1000011
D	0 1000100	1 1000100
…	…	…

  - block parity, checksum, CRC, ..

Parallel Parity

- 패리티를 블록 데이터에 적용해서 가로와 세로 데이터들에 대해서 패리티를 적용하면 에러를 검출하여 그 위치를 찾아 정정할 수 있다.

1	0	1	0	1	1	1	1	*0*
1	0	0	0	0	0	1	1	*1*
0	1	0	0	0	0	0	0	*1*
1	1	1	1	0	0	0	0	*0*
1	0	1	1	1	0	0	1	*1*
0	0	0	0	0	1	1	1	*1*
1	1	1	1	1	1	1	1	*0*
0	1	1	1	1	0	0	0	*0*
1	*0*	*1*	*0*	*0*	*1*	*0*	*1*	*0*

원래 데이터 블록

1	0	1	0	1	1	1	1	*0*
1	0	0	0	0	0	1	1	*1*
0	1	0	0	0	0	0	0	*1*
1	1	1	1	0	0	0	0	*0*
1	0	1	1	0	0	0	1	*1*
0	0	0	0	0	1	1	1	*1*
1	1	1	1	1	1	1	1	*0*
0	1	1	1	1	0	0	0	*0*
1	*0*	*1*	*0*	*0*	*1*	*0*	*1*	*0*

에러가 발생한 블록

오류 탐지 및 수정(error detection and correction)

- **Hamming code**, 어떤 r >= 2 인 정수에 대하여

$$2^r \geq r + k + 1$$

- $n = 2^r - 1$, $k = 2^r - 1 - r = n - r$, 을 만족하는 n, k 에 대해서
- 이를 (n, k)Hamming code라고 하고 R = k/n 을 Hamming rate라고 한다. 여기서 n은 총 메시지 길이이며, k는 순수 메시지 길이이다
- 예를 들어 총 메시지 비트 수 n = 7이라면 r=3 이 되어 순수 메시지 길이 k = 4 이다. 이는 4비트 데이터를 Hamming code화 하려면 3비트의 parity 비트가 추가되어 총 길이는 7비트가 된다는 것을 의미한다.
- 해밍코드에서는 **짝수 패리티**를 사용

Hamming Code 생성

- 임의의 비트 수를 가지는 정보 J에 대한 Single-Error Correcting (SEC) code를 생성하는 알고리즘
  - J를 구성하는 비트들에 대해 1부터 번호를 매긴다
    - bit 1, 2, 3, 4, 5, .., etc.
- 비트에 매긴 번호를 2진수로 변환한다
  - 1, 10, 11, 100, 101, etc
- 2의 지수승 번호로 표현되는 위치의 모든 비트는 parity 비트가 된다 : 1, 2, 4, 8, etc. (1, 10, 100, 1000, … etc)
  - 즉, P1, P2, P4, P8, …, etc로 even party로 생성된다
  - 생성 알고리즘 참고
- 2의 지수승이 아닌 번호로 표현되는 위치의 비트는 데이터 비트이다
  - D3, D5, D6, D7, D9, …, etc

Even Parity 기반의 Parity Bit 생성 알고리즘

- Parity 비트 P1 = XOR { bit-1 (the parity bit itself), 3, 5, 7, 9, …} → LSB가 1인 모든 비트들을 XOR
- Parity 비트 P2 = XOR { bit-2 (the parity bit itself), 3, 6, 7, 10, 11, …} → 2 번째 LSB가 1인 모든 비트들의 XOR
- Parity 비트 P4 = XOR { bit-4 부터 –7, 12–15, 20–23, … } → 3 번째 LSB가 1인 모든 비트들의 XOR
- Parity 비트 P8 = XOR { bits 8–15, 24–31, 40–47, … } → 4 번째 LSB가 1인 모든 비트들의 XOR
- .
- .

비트위치	1	2	3	4	5	6	7	8	9	10	11	12
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
P_1 영역	✓		✓		✓		✓		✓		✓	
P_2 영역		✓	✓			✓	✓			✓	✓	
P_4 영역				✓	✓	✓	✓					✓
P_8 영역								✓	✓	✓	✓	✓

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	X		X		X		X		X		X		X		X		X		X		. . .
	p2		X	X			X	X			X	X			X	X			X	X		
	p4				X	X	X	X					X	X	X	X					X	
	p8								X	X	X	X	X	X	X	X						
	p16																X	X	X	X	X	

8비트 데이터의 Hamming Code

$$P_1 = D_3 \oplus D_5 \oplus D_7 \oplus D_9 \oplus D_{11}$$
$$P_2 = D_3 \oplus D_6 \oplus D_7 \oplus D_{10} \oplus D_{11}$$
$$P_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_{12}$$
$$P_8 = D_9 \oplus D_{10} \oplus D_{11} \oplus D_{12}$$

D_3	D_5	D_6	D_7	D_9	D_{10}	D_{11}	D_{12}
0	0	1	0	1	1	1	0

$$P_1 = D_3 \oplus D_5 \oplus D_7 \oplus D_9 \oplus D_{11} = 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 0$$
$$P_2 = D_3 \oplus D_6 \oplus D_7 \oplus D_{10} \oplus D_{11} = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1$$
$$P_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_{12} = 0 \oplus 1 \oplus 0 \oplus 0 = 1$$
$$P_8 = D_9 \oplus D_{10} \oplus D_{11} \oplus D_{12} = 1 \oplus 1 \oplus 1 \oplus 0 = 1$$

❖ 해밍코드에서 패리티 비트 생성 과정

비트위치	1	2	3	4	5	6	7	8	9	10	11	12
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
원본 데이터			0		0	1	0		1	1	1	0
P_1 영역	0		0		0		0		1		1	
P_2 영역		1	0			1	0			1	1	
P_4 영역				1	0	1	0					0
P_8 영역								1	1	1	1	0
생성된 코드	0	1	0	1	0	1	0	1	1	1	1	0

- **해밍코드에서 패리티 비트 검사 과정**
- 전송된 데이터 : 0101110011110 (12 bit)

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
0	1	0	1	1	1	0	1	1	1	1	0

$$P_1 = P_1 \oplus D_3 \oplus D_5 \oplus D_7 \oplus D_9 \oplus D_{11} = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1$$

$$P_2 = P_2 \oplus D_3 \oplus D_6 \oplus D_7 \oplus D_{10} \oplus D_{11} = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0$$

$$P_4 = P_4 \oplus D_5 \oplus D_6 \oplus D_7 \oplus D_{12} = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 1$$

$$P_8 = P_8 \oplus D_9 \oplus D_{10} \oplus D_{11} \oplus D_{12} = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

- 검사된 패리티를 P_8 P_4 P_2 P_1 순서대로 정렬한다.
- 모든 패리티가 0이면 에러가 없는 것이고, 그렇지 않으면 에러가 발생한 것이다.
- 결과가 0101이므로 에러가 있으며, 이것을 10진수로 바꾸면 5가 된다. 즉, 수신된 데이터 010111011110에서 앞에서 5번째 비트 1이 에러가 발생한 것이므로 010101011110으로 바꾸어 주면 에러가 정정된다.

❑ 해밍코드에서 에러가 발생한 경우 교정

비트위치		1	2	3	4	5	6	7	8	9	10	11	12
기호		P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
Error 해밍코드		0	1	0	1	1	1	0	1	1	1	1	0
P_1 계산	1	0		0		1		0		1		1	
P_2 계산	0		1	0			1	0			1	1	
P_4 계산	1				1	1	1	0					0
P_8 계산	0								1	1	1	1	0

P_8 P_4 P_2 P_1 =0101 = ***5*** : 5번 비트에 에러가 발생. 1 → 0으로 교정

2진 논리(Binary Logic)

- Boolean Algebra(부울대수)를 따른다
- 진리표(truth table)를 사용하여 동작이 표현될 수 있다
- 3가지 기본 논리 연산
  - AND, OR, NOT

AND

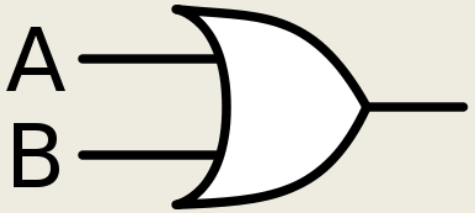
- 보통 '•'을 사용한다

Truth table for A • B

A	B	A • B
0	0	0
0	1	0
1	0	0
1	1	1

OR

- 보통 ' + '를 사용하여 표현



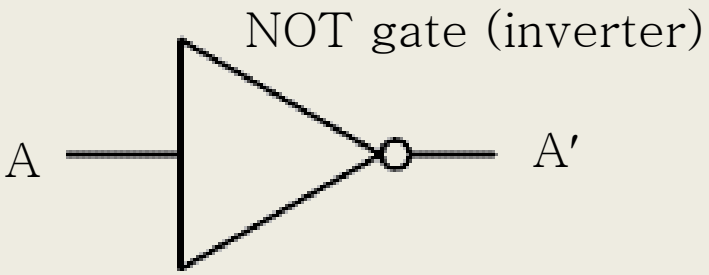
OUT = A + B

Truth table for A + B

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

NOT

- 보통 ′ 혹은 상단에 – 을 붙여 표현한다
- 예를 들어 A′ 혹은 $\overline{A}$

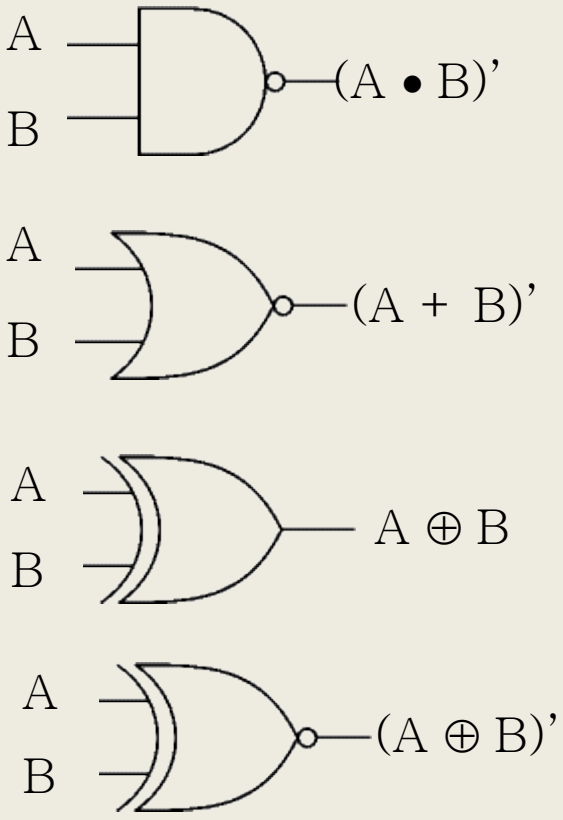


Truth table for A′

A	A′
0	1
1	0

기타 주요 논리 연산

- **NAND (NOT AND), NOR(NOT NOR), XOR, XNOR**



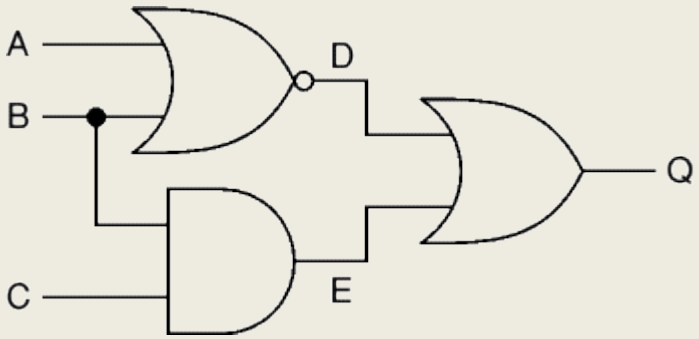
0	0	1
0	1	1
1	0	1
1	1	0

0	0	1
0	1	0
1	0	0
1	1	0

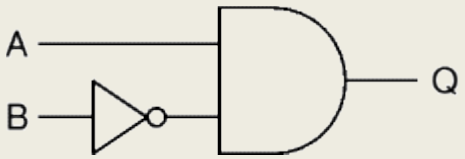
0	0	0
0	1	1
1	0	1
1	1	0

0	0	1
0	1	0
1	0	0
1	1	1

- 예



D = (A + B)’
E = B • C
Q = D + E = ((A + B)’) + (B • C)



Q = A • B ′

Boolean Algebra

- (1a) $x \cdot y = y \cdot x$
- (1b) $x + y = y + x$
- (2a) $x \cdot (y \cdot z) = (x \cdot y) \cdot z$: multi-input AND
- (2b) $x + (y + z) = (x + y) + z$: multi-input OR
- (3a) $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
- (3b) $x + (y \cdot z) = (x + y) \cdot (x + z)$
- (4a) $x \cdot x = x$
- (4b) $x + x = x$
- (5a) $x \cdot (x + y) = x$
- (5b) $x + (x \cdot y) = x$
- (6a) $x \cdot x' = 0$
- (6b) $x + x' = 1$
- (7) $(x')' = x$
- (8a) $(x \cdot y)' = x' + y'$: De Morgan's Theorem
- (8b) $(x + y)' = x' \cdot y'$: De Morgan's Theorem

문자형

- 문자도 숫자를 이용하여 표현
- 공통적인 규격이 필요하다.
- 아스키 코드(**ASCII:** American Standard Code for Information Interchange)
  - 8비트를 사용하여 영어 알파벳 표현
  - (예) !는 33, 'A'는 65, 'B'는 66, 'a'는 97, 'b'는 98

```
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

- 그 외에 EBCIDIC, 표준 BCD 등 다양한 코드가 존재한다

표준 ASCII 코드표

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

코드의 구성

b_9	b_8 b_7 b_6 b_5	b_4 b_3 b_2 b_1
패리티	존(zone)	디지트(digit)
1	4	4

b_8 b_7		b_6 b_5	
0 0	통신제어문자		
0 1	특수문자		
1 0	소문자	0 0	a ~i
		0 1	j~r
		1 0	s~z
		1 1	
1 1	대문자/숫자	0 0	A~I
		0 1	J~R
		1 0	S~Z
		1 1	0~9

16진		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	2진	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	NUL	SOH	STX	ETX		HT		DEL				VT	FF	CR	SO	SI
1	0001	DLE						BS		CAN	EM			IFS	IGS	IRS	IUS
2	0010						LF	ETB	ESC						ENQ	ACK	BEL
3	0011			SYN					EOT						NAK		SUB
4	0100	space										[	.		(	+	|
5	0101	&										!	$	*	)		^
6	0110	-	/									|	,	%	_	>	?
7	0111										`	:	#	@	'	=	"
8	1000		a	b	c	d	e	f	g	h	i						
9	1001		j	k	l	m	n	o	p	q	r						
A	1010		~	s	t	u	v	w	x	y	z						
B	1011																
C	1100	{	A	B	C	D	E	F	G	H	I						
D	1101	}	J	K	L	M	N	O	P	Q	R						
E	1110	₩		S	T	U	V	W	X	Y	Z						
F	1111	0	1	2	3	4	5	6	7	8	9						

문자 변수

- char형의 변수에 문자 저장

```
char c;
char answer;
char code;
```

- char형의 변수에 문자를 저장하려면 아스키 코드 값을 대입

```
code = 65;          // 'A' 저장
code = 'A';
```

예제

```c
/* 문자 변수와 문자 상수*/
#include <stdio.h>

int main(void)
{
	char code1 = 'A';	// 문자 상수로 초기화
	char code2 = 65;	// 아스키 코드로 초기화

	printf("문자 상수 초기화 = %c\n", code1);
	printf("아스키 코드 초기화 = %c\n", code2);
}
```

```
문자 상수 초기화 = A
아스키 코드 초기화 = A
```

(Q) 1과 '1'의 차이점은?

(A) 1은 정수 상수이고 '1'은 문자 상수이다.

제어 문자(Control Characters)

- 인쇄 목적이 아니라 제어 목적으로 사용되는 문자들
  - (예) 줄바꿈 문자, 탭 문자, 벨소리 문자, 백스페이스 문자
- 제어 문자를 나타내는 방법
  - 아스키 코드를 직접 사용

```
char beep = 7;
printf("%c", beep);
```

  - 이스케이프 시퀀스 사용

```
char beep = '\a';
printf("%c", beep);
```

Escape 시퀀스

제어 문자 이름	제어 문자 표기	값	의미
널문자	₩0	0	문자열의 끝을 표시
경고(bell)	₩a	7	"삐"하는 경고 벨소리 발생
백스페이스(backspace)	₩b	8	커서를 현재의 위치에서 한 글자 뒤로 옮긴다.
수평탭(horizontal tab)	₩t	9	커서의 위치를 현재 라인에서 설정된 다음 탭 위치로 옮긴다.
줄바꿈(newline)	₩n	10	커서를 다음 라인의 시작 위치로 옮긴다.
수직탭(vertical tab)	₩v	11	설정되어 있는 다음 수직 탭 위치로 커서를 이동
폼피드(form feed)	₩f	12	주로 프린터에서 강제적으로 다음 페이지로 넘길 때 사용된다.
캐 리 지 리 턴 (carriage return)	₩r	13	커서를 현재 라인의 시작 위치로 옮긴다.
큰따옴표	₩"	34	원래의 큰따옴표 자체
작은따옴표	₩'	39	원래의 작은따옴표 자체
역슬래시(back slash)	₩₩	92	원래의 역슬래시 자체

예제

```c
/* 이스케이프 시퀀스 */
#include <stdio.h>

int main(void)
{
	printf("이스케이프 시퀀스는 \\와 의미를 나타내는 글자를 붙여서 기술\n");
	printf("'\\a'는 경고를 나타내는 제어문자이다. \n");
	printf("'\\007'로도 표현이 가능하다. \n");
	printf("경고를 출력해 보자'\\007'을 출력한다\007 \n");

}
```

```
이스케이프 시퀀스는 ₩와 의미를 나타내는 글자를 붙여서 기술
'₩a'는 경고를 나타내는 제어 문자이다.
'₩007'로도 표현이 가능하다.
경고를 출력해보자 '₩007'을 출력한다
```