데이터 구조

자료 구조의 소개

수업 진행 정리

❖ 분반 별 인원 수 및 예상 학점 인원수

- 오전반(2분반) - 34명 - 10등(A), 23등(B)
- 오후반(1분반) - 29명 - 8등(A), 20등(B)

❖ 팀플 관련

- 오전반 - 1~4명 (약 17문제 예상)
- 오후반 - 무조건 2명 (15문제 예상)
- 프로그램 공유 및 같이 공부하기 : 표절률 50% 전체 모두 가능
- 팀원명 25일까지 제출(메일로)

Contents

❖ 학습목표

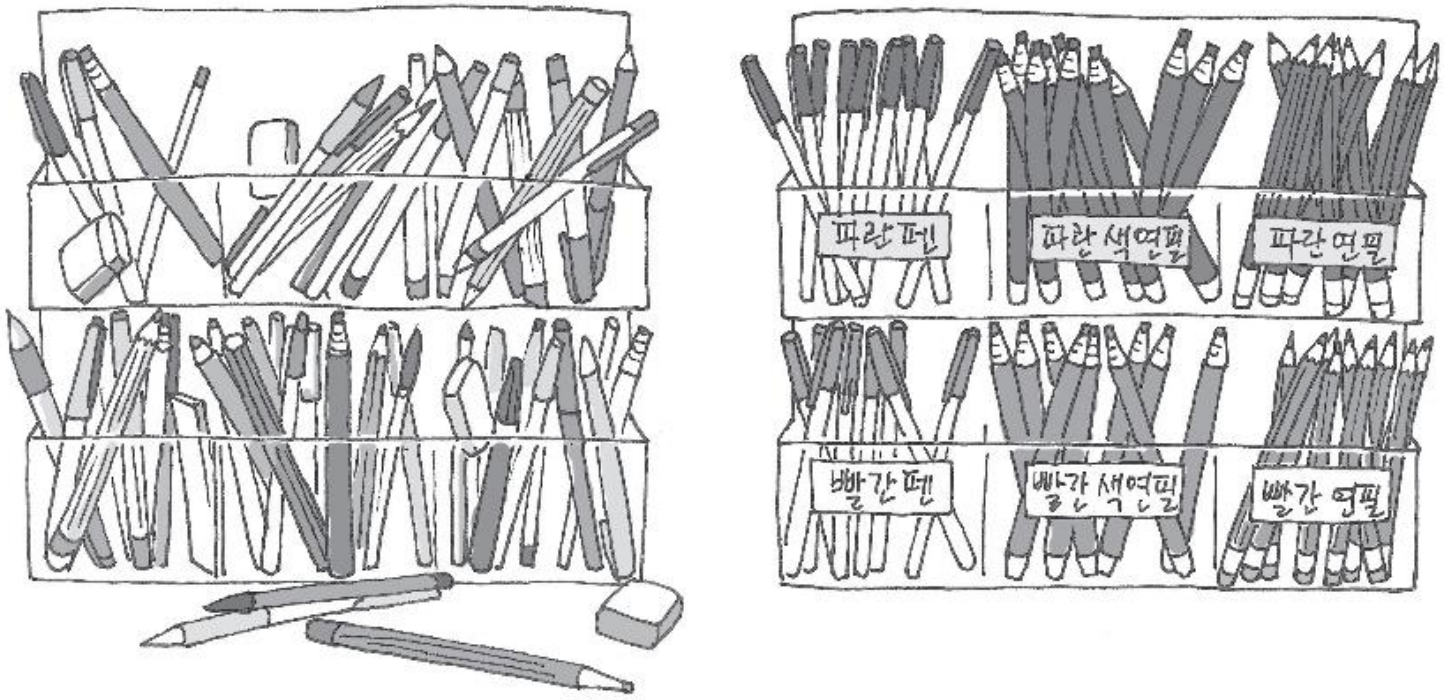
- 자료구조의 의미와 중요성을 알아본다.
- 자료구조에서 다루는 내용을 알아본다.
- 컴퓨터 내부의 2진수 코드 체계를 알아본다.
- 자료 형태에 따른 자료 표현 형식을 알아본다.
- 자료를 추상화하고 구체화하는 개념을 이해한다.
- 알고리즘의 개념을 이해하고 알고리즘의 표현 방법을 알아본다.
- 알고리즘의 성능 분석 방법을 알아본다.

❖ 내용

1. 자료구조의 이해 : 개념

❖ 자료구조의 개념

- 자료를 효율적으로 표현하고 저장하고 처리할 수 있도록 정리하는 것



(a) 자료구조를 적용하기 전

(b) 자료구조를 적용한 후

그림 1-1 생활 속에서 자료구조를 적용한 예

1. 자료구조의 이해 : 개념

❖ 컴퓨터 분야에서 자료구조를 왜 배워야 하는가?

- 컴퓨터가 효율적으로 문제를 처리하기 위해서는 문제를 정의하고 분석하여 그에 대한 최적의 프로그램을 작성해야 한다.
  - 자료구조에 대한 개념과 활용 능력 필요!

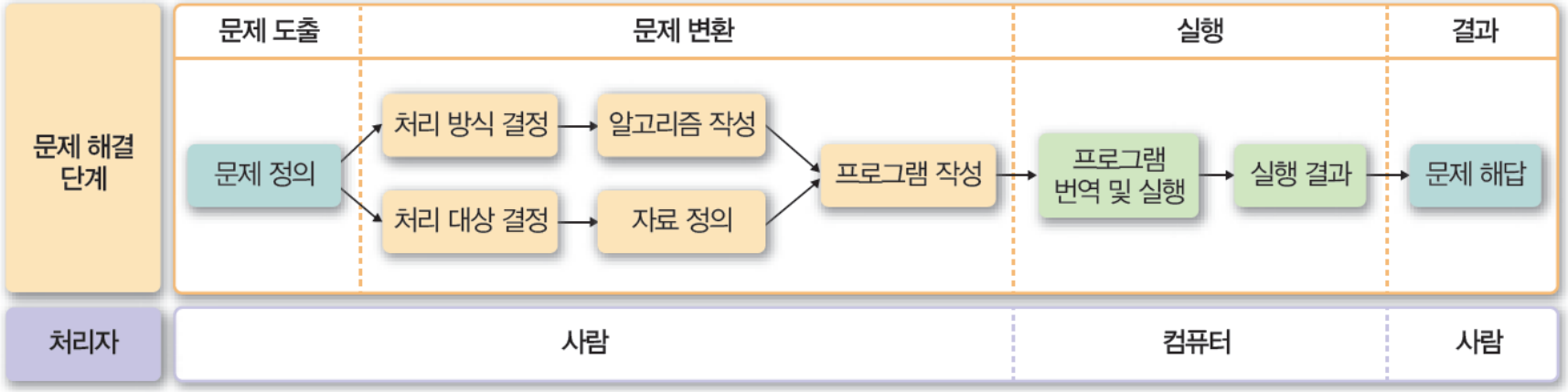


그림 1-2 문제 해결 과정

1. 자료구조의 이해 : 분류

❖ 자료의 형태에 따른 분류

- 단순 구조
  - 정수, 실수, 문자, 문자열, 등의 기본 자료형
- 선형 구조
  - 자료들 사이의 관계가 1:1 관계
  - 순차 리스트, 연결 리스트, 스택, 큐, 데크 등
- 비선형 구조
  - 자료들 사이의 관계가 1:다, 또는 다:다 관계
  - 트리, 그래프 등
- 파일 구조
  - 서로 관련 있는 필드로 구성된 레코드의 집합인 파일에 대한 구조
  - 순차 파일, 색인 파일, 직접 파일 등

1. 자료구조의 이해 : 분류

❖ 자료의 형태에 따른 분류와 이 책에서 다루는 세부 주제

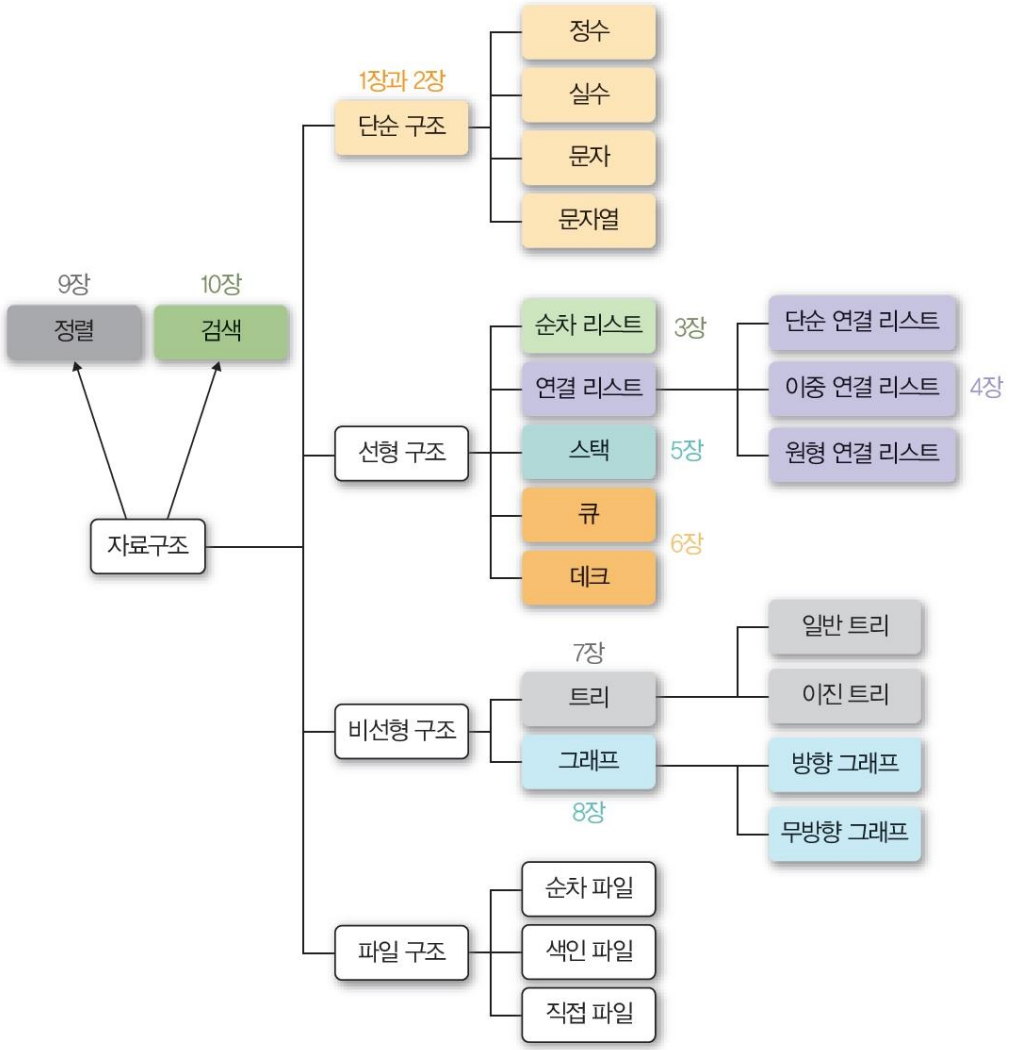


그림 1-4 자료구조의 형태에 따른 분류와 이 책에서 다루는 세부 주제

2. 자료의 표현

❖ 컴퓨터에서의 자료 표현

- 숫자, 문자, 그림, 소리, 기호 등 모든 형식의 자료를 2진수 코드로 표현하여 저장 및 처리
- 2진수 코드란?
  - 1과 0, On과 Off, 참True과 거짓False의 조합
- 2진수 코드의 단위

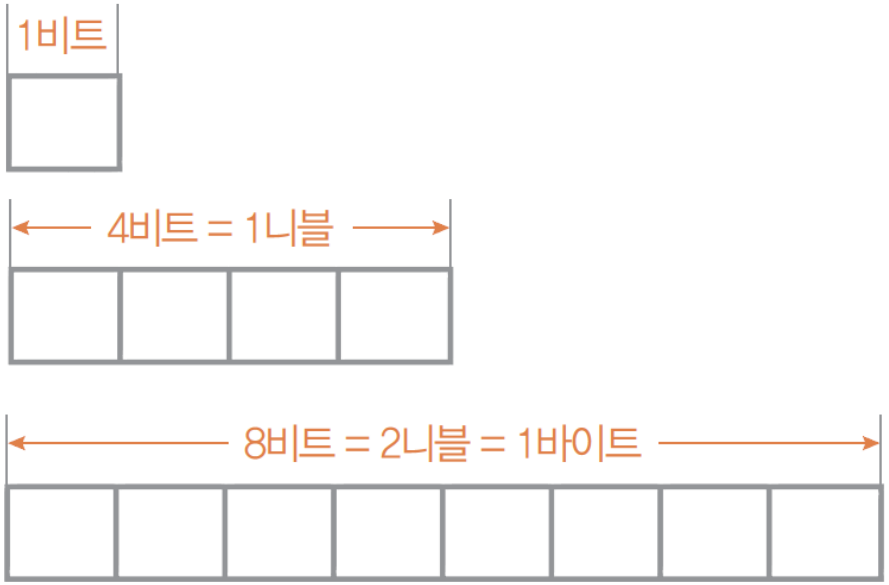
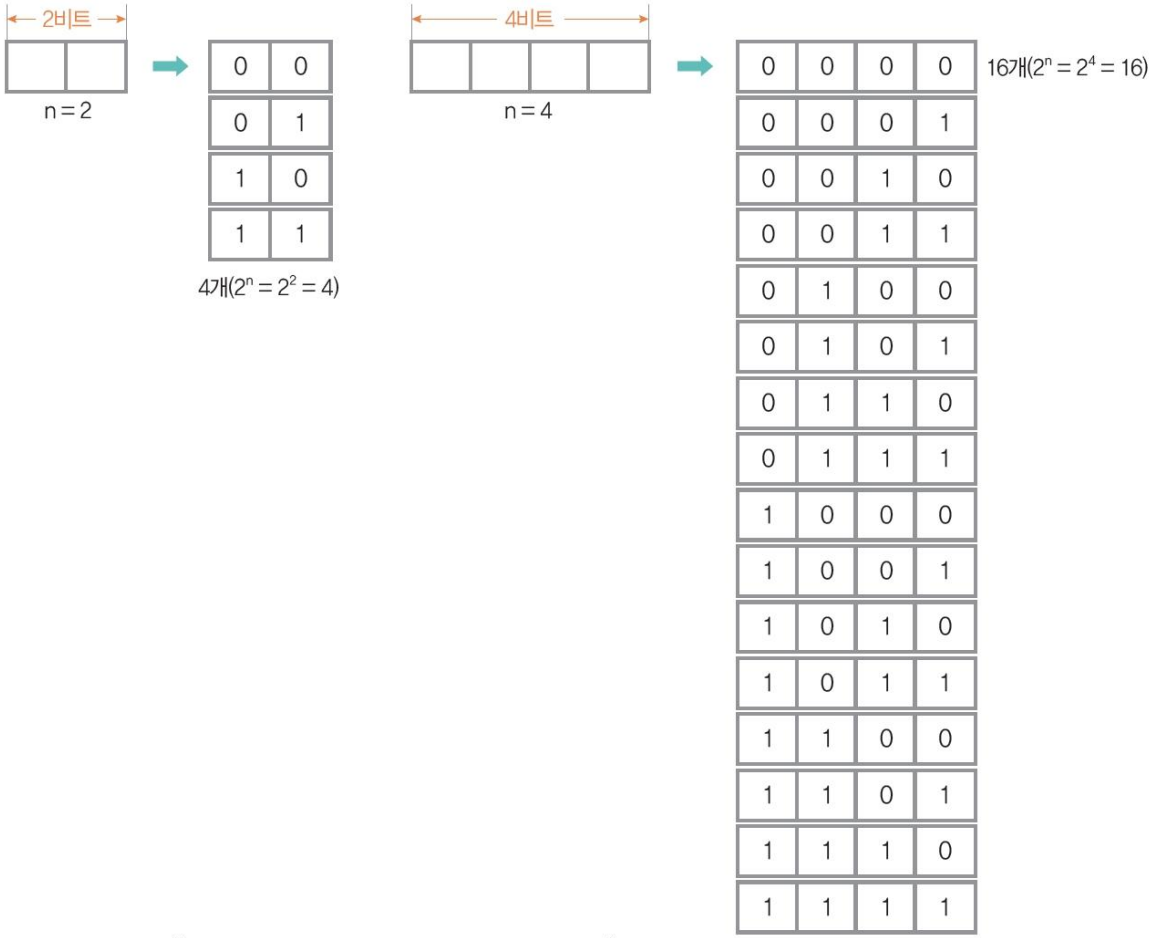


그림 1-5 컴퓨터의 자료 표현 : 비트, 니블, 바이트

2. 자료의 표현

❖ 디지털 시스템에서의 자료 표현

- n개의 비트로 2^n개의 상태 표현



0	0
0	1
1	0
1	1

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

(a) n=2인 경우 : 2^2개의 상태 표현 (b) n=4인 경우 : 2^4개의 상태 표현

그림 1-6 자료 표현 예 : n개의 비트로 2^n개의 상태 표현

2. 자료의 표현

❖ 컴퓨터 내부에서 표현할 수 있는 자료의 종류

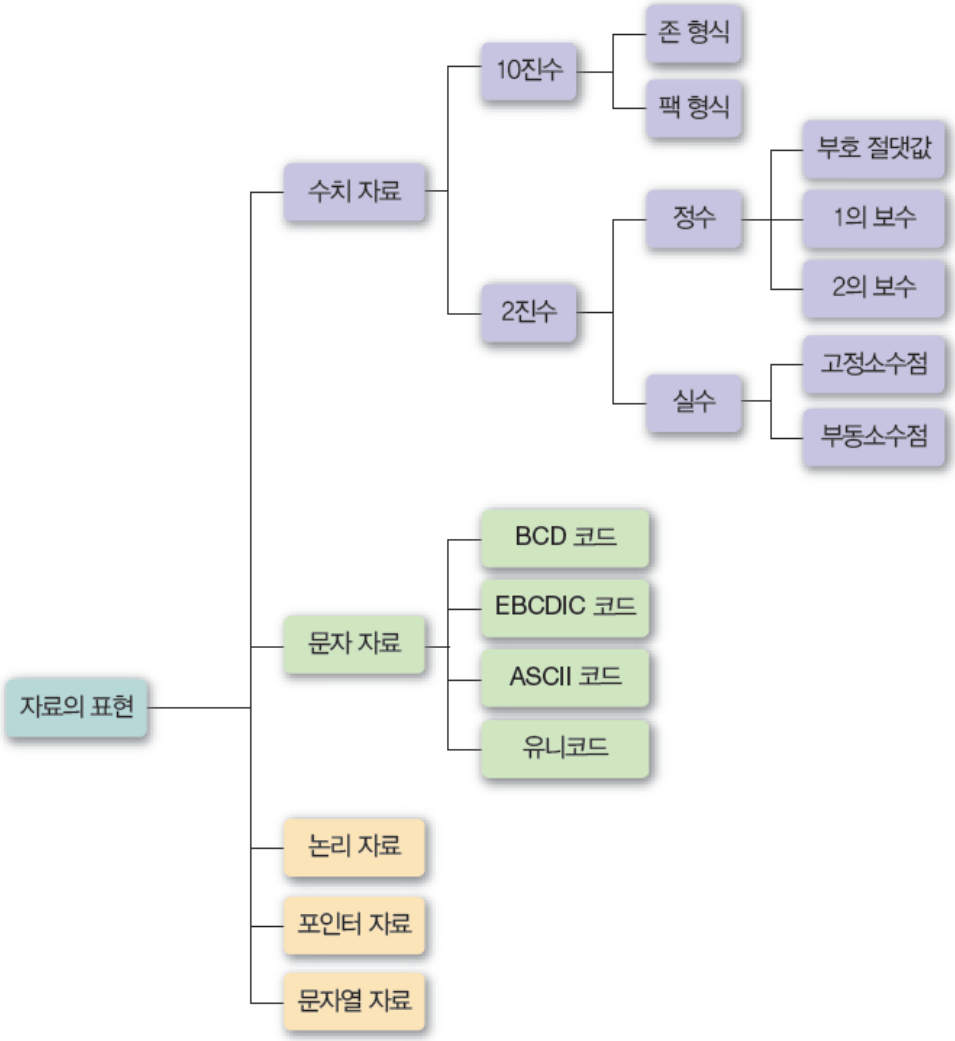


그림 1-7 컴퓨터 내부에서 자료를 표현하는 방법

2. 자료의 표현 : 수치 자료의 표현

❖ 10진수의 표현

- 존Zone 형식의 표현
  - 10진수 한 자리를 표현하기 위해서 1바이트(8비트)를 사용하는 형식
  - 존 영역
    - 상위 4비트
    - 1111로 표현
  - 수치 영역
    - 하위 4비트
    - 표현하고자 하는 10진수 한 자리 값에 대한 2진수 값을 표시
  - 존 형식의 구조

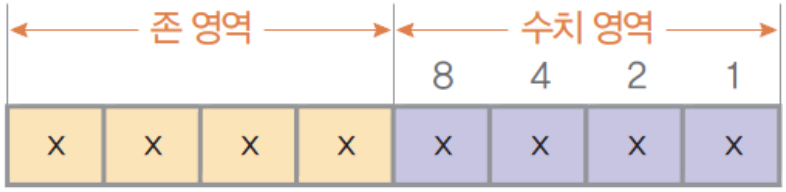


그림 1-8 존 형식의 구조

2. 자료의 표현 : 수치 자료의 표현

❖ 10진수의 표현

- 수치 영역의 값 표현 : [표 1-1]

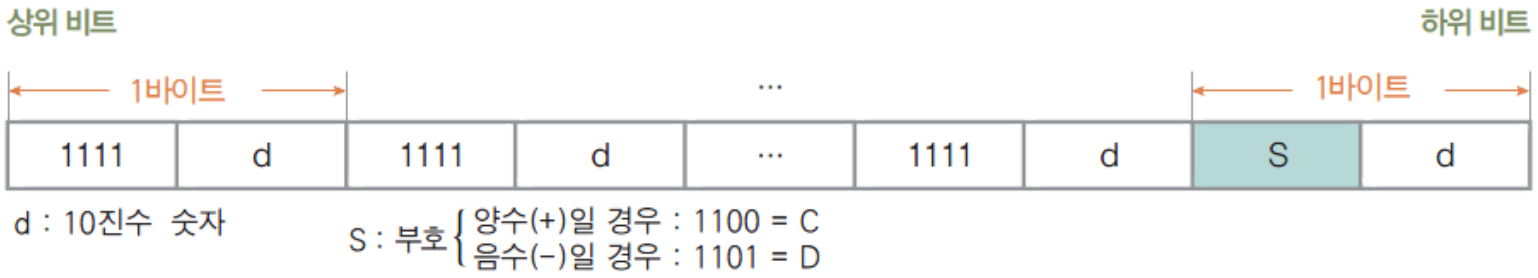
표 1-1 4비트의 2진수에 대한 10진수 표현

4비트의 2진수				10진수 변환	10진수
0	0	0	0	$0\times2^3+0\times2^2+0\times2^1+0\times2^0$	0
0	0	0	1	$0\times2^3+0\times2^2+0\times2^1+1\times2^0$	1
0	0	1	0	$0\times2^3+0\times2^2+1\times2^1+0\times2^0$	2
0	0	1	1	$0\times2^3+0\times2^2+1\times2^1+1\times2^0$	3
0	1	0	0	$0\times2^3+1\times2^2+0\times2^1+0\times2^0$	4
0	1	0	1	$0\times2^3+1\times2^2+0\times2^1+1\times2^0$	5
0	1	1	0	$0\times2^3+1\times2^2+1\times2^1+0\times2^0$	6
0	1	1	1	$0\times2^3+1\times2^2+1\times2^1+1\times2^0$	7
1	0	0	0	$1\times2^3+0\times2^2+0\times2^1+0\times2^0$	8
1	0	0	1	$1\times2^3+0\times2^2+0\times2^1+1\times2^0$	9
1	0	1	0	$1\times2^3+0\times2^2+1\times2^1+0\times2^0$	10 = A
1	0	1	1	$1\times2^3+0\times2^2+1\times2^1+1\times2^0$	11 = B
1	1	0	0	$1\times2^3+1\times2^2+0\times2^1+0\times2^0$	12 = C
1	1	0	1	$1\times2^3+1\times2^2+0\times2^1+1\times2^0$	13 = D
1	1	1	0	$1\times2^3+1\times2^2+1\times2^1+0\times2^0$	14 = E
1	1	1	1	$1\times2^3+1\times2^2+1\times2^1+1\times2^0$	15 = F

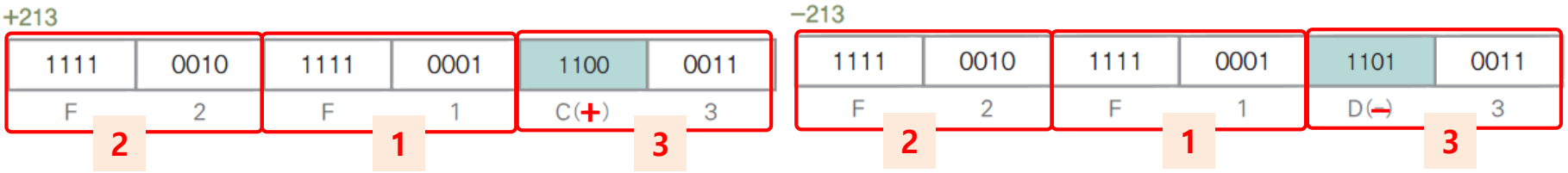
2. 자료의 표현 : 수치 자료의 표현

❖ 10진수의 표현

- 여러 자리의 10진수를 표현하는 방법
  - 10진수의 자릿수만큼 존 형식을 연결하여 사용
  - 마지막 자리의 존 영역에 부호를 표시
    - 양수(+) : 1100
    - 음수(-) : 1101



(a) 존 형식의 10진수 표현 형식



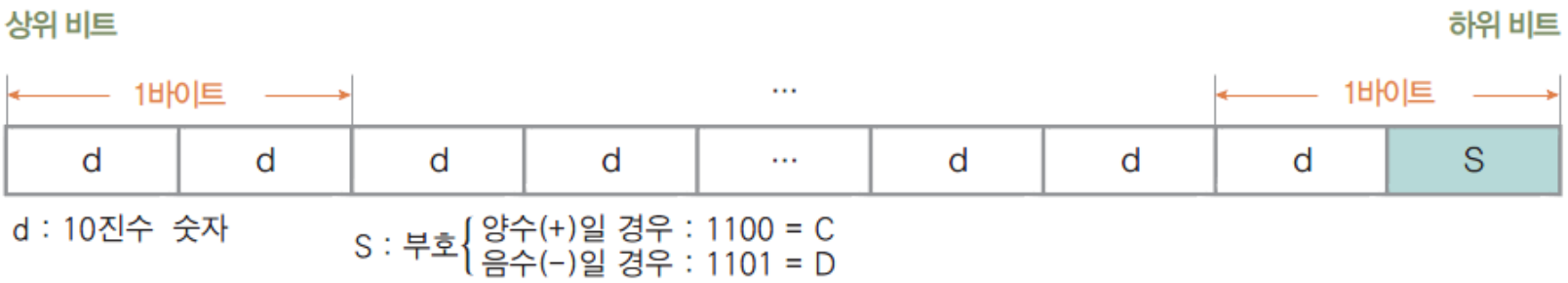
(b) 존 형식의 10진수 표현 예

그림 1-9 존 형식의 10진수 표현 형식과 예

2. 자료의 표현 : 수치 자료의 표현

❖ 10진수의 표현

- 팩Pack 형식의 표현
  - 10진수 한 자리를 표현하기 위해서 존 영역 없이 4비트를 사용하는 형식
  - 최하위 4비트에 부호를 표시
    - 양수(+) : 1100
    - 음수(-) : 1101



(a) 팩 형식의 10진수 표현 형식

(b) 팩 형식의 10진수 표현 예

그림 1-10 팩 형식의 10진수 표현 형식과 예

2. 자료의 표현 : 수치 자료의 표현

❖ 2진수의 정수 표현

- n비트의 부호 절댓값 형식
  - 최상위 1비트 : 부호 표시
    - 양수(+) : 0
    - 음수(-) : 1
  - 나머지 n-1 비트 : 이진수 표시
  - 1바이트를 사용하는 부호 절댓값 표현 예

그림 1-11 +21과 −21의 부호와 절댓값 표현 예

2. 자료의 표현 : 수치 자료의 표현

❖ 2진수의 정수 표현

- 1의 보수1' Complement 형식
  - 음수 표현에서 부호 비트를 사용하는 대신 1의 보수를 사용하는 방법
  - 음수를 n비트의 1의 보수로 만드는 방법
    - 방법 a) (2^n-1)에서 음수x (-x)의 절댓값을 뺌 ☞ (2^n-1) - |-x|
    - 방법 b) n비트를 모두 1로 만든 이진수에서 음수x (-x)의 절댓값 이진수 |-x|를 뺌
    - 방법 c) 음수x (-x)의 절댓값 이진수의 각 비트를 반대값(1→0, 0→1)으로 변환

    - 예) 10진수 음수21 (-21)을 1바이트의 1의 보수로 표현하기

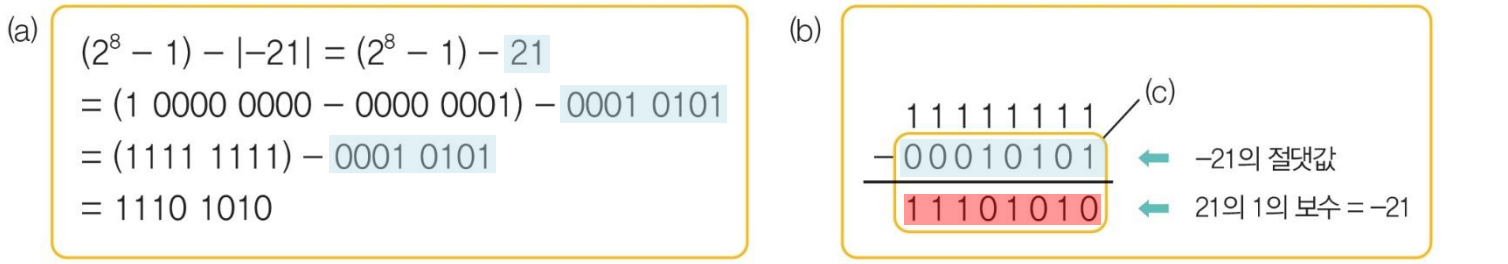


그림 1-12 1의 보수 형식 표현 과정

그림 1-13 +21과 –21의 1의 보수 형식 표현 예

2. 자료의 표현 : 수치 자료의 표현

❖ 2진수의 정수 표현

- 2의 보수2' Complement 형식
  - 음수 표현에서 부호 비트를 사용하는 대신 2의 보수를 사용하는 방법
  - 음수를 n비트의 2의 보수로 만드는 방법
    - 방법 a) 2^n에서 음수x (-x)의 절댓값을 뺌 ☞ 2^n - |-x|
    - 방법 b) n비트를 모두 1로 만든 이진수에서 음수x (-x)의 절댓값 이진수 |-x|를 뺀 값(1의 보수)에 1을 더함
    - 예) 10진수 음수21 (-21)을 1바이트의 2의 보수로 표현하기

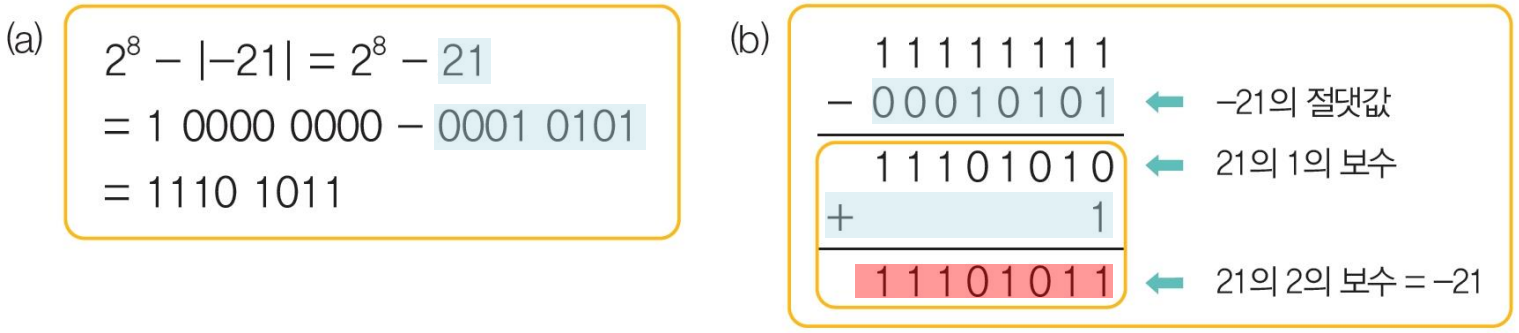


그림 1-14 2의 보수 형식 표현 과정

2. 자료의 표현 : 수치 자료의 표현

❖ 2진수의 정수 표현

- 2의 보수2' Complement 형식
  - 1바이트를 사용하는 2의 보수 형식의 예

그림 1-15 +21과 −21의 2의 보수 형식 표현 예

- 2진수 정수의 세 가지 표현 방법에서 양수의 표현은 같고 **음수의 표현만 다르다.**

2. 자료의 표현 : 수치 자료의 표현

표 1-3 2진수를 표현하는 세 가지 방법 비교

표현 방법	특징
부호와 절댓값 형식	• MSB값을 바꿔 음수를 간단히 표현할 수 있다. • 가산기와 감산기가 모두 필요하므로 하드웨어 구성 비용이 많이 든다. • +0(00000000)과 −0(10000000)이 존재하므로 논리적으로 맞지 않다. • n비트로 $-(2^{n-1}-1) \sim +(2^{n-1}-1)$의 범위를 표현할 수 있다.
1의 보수 형식	• (A−B) 뺄셈을 (A+(B의 1의 보수))로 변환하여 계산할 수 있으므로 가산기 회로로 감산을 수행할 수 있다. • +0(00000000)과 −0(11111111)이 존재하므로 논리적으로 맞지 않다. • n비트로 $-(2^{n-1}-1) \sim +(2^{n-1}-1)$의 범위를 표현할 수 있다.
2의 보수 형식	• (A−B) 뺄셈을 (A+(B의 2의 보수))로 변환하여 계산할 수 있으므로 가산기 회로로 감산을 수행할 수 있다. • 덧셈 연산에서 발생하는 오버플로 처리가 1의 보수 형식보다 간단하다. • 컴퓨터 시스템에서 실제로 사용하는 형식이다. • n비트로 $-2^{n-1} \sim +(2^{n-1}-1)$의 범위를 표현할 수 있다.

2. 자료의 표현 : 수치 자료의 표현

❖ 2진수의 실수 표현

- 고정 소수점 표현
  - 소수점이 항상 최상위 비트의 왼쪽 밖에 고정되어 있는 것으로 취급하는 방법
  - 고정 소수점 표현의 **00010101**은 **0.00010101**의 실수 값을 의미

- 부동 소수점 형식의 표현
  - 고정 소수점 형식에 비해서 표현 가능한 값의 범위가 넓음
  - 실수를 구분하여 표현

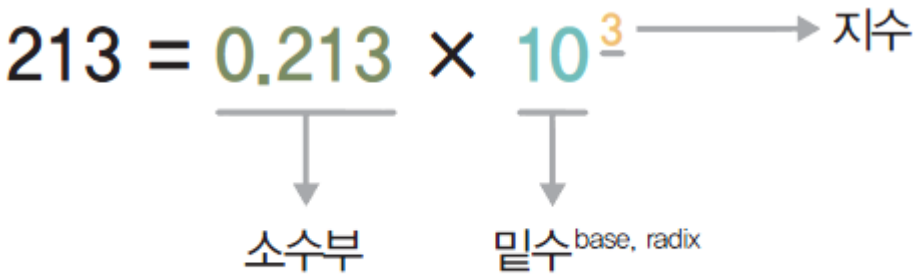


그림 1-16 실수의 과학적 표기

2. 자료의 표현 : 수치 자료의 표현

❖ 2진수의 실수 표현

- 부동 소수점 형식의 표현

4바이트를 사용하는 부동 소수점 형식

8바이트를 사용하는 부동 소수점 형식

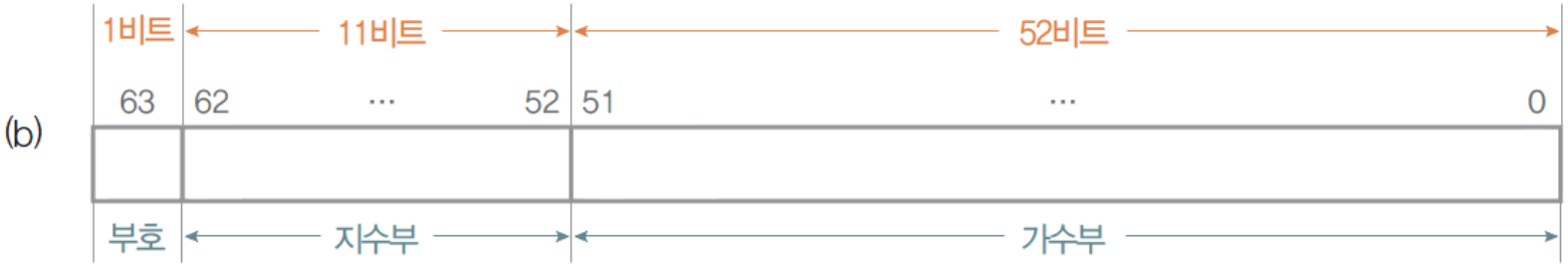


그림 1-17 부동소수점 표현 형식

2. 자료의 표현 : 문자 자료의 표현

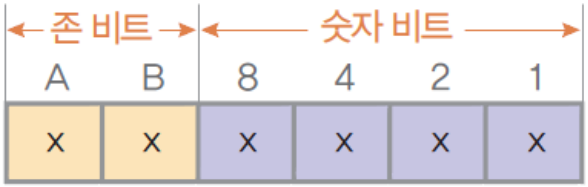
❖ 문자 자료의 표현

- 문자에 대한 이진수 코드를 정의하여 사용
- 문자에 대한 이진수 코드표
  - BCD 코드
  - EBCDIC 코드
  - ASCII 코드

2. 자료의 표현 : 문자 자료의 표현

❖ BCD 코드

- 6비트를 사용하여 문자 표현
  - 상위 2비트 : 존 비트
  - 하위 4비트 : 2진수 비트
  - 존 비트와 2진수 비트를 조합하여 10진수 0~9와 영어 대문자, 특수 문자를 표현



존 비트 AB의 값
- 00 : 숫자 0, 1~9(1010, 0001~1001)
- 01 : 문자 A~I(0001~1001)
- 10 : 문자 J~R(0001~1001)
- 11 : 문자 S~Z(0010~1001)

그림 1-19 BCD 코드의 구성

2. 자료의 표현 : 문자 자료의 표현

❖ BCD 코드

존 비트 01과 숫자 비트 0001을 연결한 01 0001이 A에 대한 BCD 코드

표 1-4 BCD 코드 표

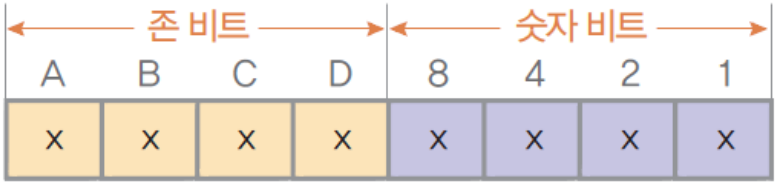
존 비트	숫자 비트	표현 문자	존 비트	숫자 비트	표현 문자	존 비트	숫자 비트	표현 문자	존 비트	숫자 비트	표현 문자
00	0001	1	01	0001	A	10	0001	J			
00	0010	2	01	0010	B	10	0010	K	11	0010	S
00	0011	3	01	0011	C	10	0011	L	11	0011	T
00	0100	4	01	0100	D	10	0100	M	11	0100	U
00	0101	5	01	0101	E	10	0101	N	11	0101	V
00	0110	6	01	0110	F	10	0110	O	11	0110	W
00	0111	7	01	0111	G	10	0111	P	11	0111	X
00	1000	8	01	1000	H	10	1000	Q	11	1000	Y
00	1001	9	01	1001	I	10	1001	R	11	1001	Z
00	1010	0									

2. 자료의 표현 : 문자 자료의 표현

❖ EBCDIC 코드

- 8비트를 사용하여 문자 표현
  - 상위 4비트 : 존 비트
  - 하위 4비트 : 2진수 비트
  - 존 비트와 2진수 비트를 조합하여 10진수 0~9와 영어 대문자/소문자와 특수문자를 표현

- EBCDIC 코드의 구성



존 비트 AB의 값
- 00 : 여분
- 01 : 특수문자
- 10 : 영어 소문자
- 11 : 영어 대문자

존 비트 CD의 값
- 00 : 문자 A~I(0001~1001)
- 01 : 문자 J~R(0001~1001)
- 10 : 문자 S~Z(0010~1001)
- 11 : 숫자 0~9(0000~1001)

그림 1-20 EBCDIC 코드의 구성

2. 자료의 표현 : 문자 자료의 표현

❖ EBCDIC 코드

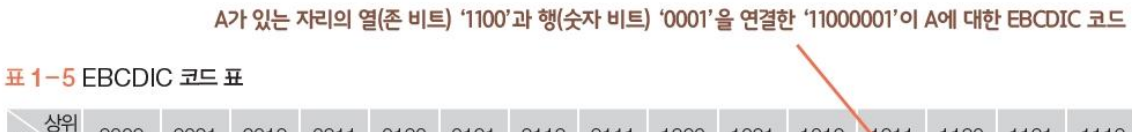


표 1-5 EBCDIC 코드 표

상위 / 하위	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL	DLE	DS		SP	&	–						{	}	₩(\)	0
0001	SOH	DC1	SOS				/		a	j	~		A	J		1
0010	STX	DC2	FS	SYN					b	k	s		B	K	S	2
0011	ETX	TM							c	l	t		C	L	T	3
0100	PF	RES	BYP	PN					d	m	u		D	M	U	4
0101	HT	NL	LF	RS					e	n	v		E	N	V	5
0110	LC	BS	ETB	UC					f	o	w		F	O	W	6
0111	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
1000	GE	CAN							h	q	y		H	Q	Y	8
1001	RLF	EM							i	r	z		I	R	Z	9
1010	SMM	CC	SM		¢	!		:								
1011	VT	CU1	CU2	CU3	.	$	,	#								
1100	FF	IFS		DC4	<	*	%	@								
1101	CR	IGS	ENQ	NAK	(	)	_	'								
1110	SO	IRS	ACK		+	;	>	=								
1111	SI	IUS	BEL	SUB	|	¬	?	"								

- EOT End Of Transmission
- ESC ESCape
- ETB End of Transmission Block
- ETX End of TeXt
- FF From Feed
- FS Field Separator
- HT Horizontal Tab
- IFS Interchange File Separator
- PN Punch oN
- RES REStore
- RS Reader Stop
- SI Shift In
- SM Set Mode
- SMM Start of Manual Message
- SO Shift Out
- SOH Start Of Heading
- IGS Interchange Group Separator
- IL IdLe
- ACK ACKnowledge
- BEL BELl
- BS BackSpace
- BYP BYPass
- CAN CANcel
- CC Cursor Control
- CR Carriage Return
- CU1 Customer Use 1
- CU3 Customer Use 3
- IRS Interchange Record Separator
- IUS Interchange Unit Separator
- LC Lower Case
- LF Line Feed
- NAK Negative AcKnowledge
- NL New Line
- NUL NULl
- PF Punch oFf
- SOS Start Of Significance
- DC1 Device Control 1
- DC2 Device Control 2
- DC4 Device Control 4
- DEL DELete
- DLE Data Link Escape
- CU2 Customer Use 2
- DS Digital Select
- EM End of Medium
- ENQ ENQuire
- SP SPace
- STX Start of TeXt
- SUB SUBstitute
- SYN SYNchronous
- TM Tape Mark
- UC Upper Case
- VT Vertical Tab
- ¢ Cent Sign
- ¬ Logical NOT

2. 자료의 표현 : 문자 자료의 표현

❖ ASCII 코드

- 7비트를 사용하여 문자 표현
  - 상위 3비트 : 존 비트
  - 하위 4비트 : 2진수 비트
  - 존 비트와 2진수 비트를 조합하여 10진수 0~9와 영어 대문자/소문자, 특수문자를 표현

- ASCII 코드의 구성

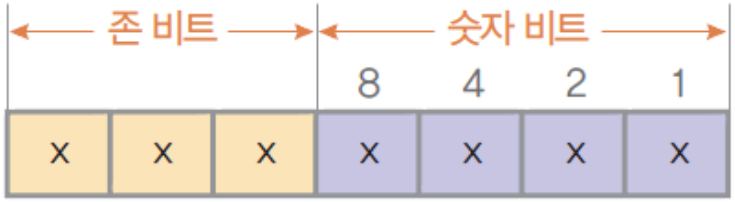


그림 1-21 ASCII 코드의 구성

2. 자료의 표현 : 문자 자료의 표현

❖ ASCII 코드

A가 있는 자리의 열(존 비트) '100'과 행(숫자 비트) '0001'을 연결한 '1000001'이 A에 대한 ASCII 코드

표 1-6 ASCII 코드 표

하위 \ 상위	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	END	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	₩(\)	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

- GS Group Separator
- RS Record Separator
- US Unit Separator

2. 자료의 표현 : 논리자료의 표현

❖ 유니코드

- EBCDIC 코드나 ASCII 코드는 최대 8비트로 숫자, 몇 가지 특수문자, 알파벳 정의하므로 문자 코드 표에 정의되어 있지 않은 문자 표현 불가능
- 이런 문제 해결 위해 세계 여러 나라의 언어를 통일된 방법으로 표현할 수 있도록 정의한 국제 표준 코드(ISO/IEC 10646)
- 2바이트를 조합하여 하나의 글자를 표현하기 때문에 1바이트 코드로 표현할 수 없었던 다양한 언어를 표현.
- 유니코드 표는 http://www.unicode.org/에서 확인 가능
- 초기 IBM 컴퓨터 시스템에서는 BCD 코드를 사용하다가 더 많은 문자 코드를 표현할 수 있는 EBCDIC코드로 대체, 그러다 미국 표준 코드인 ASCII 코드 일반화, 현재는 표현의 한계를 극복한 유니코드가 일반화
- XML, Java, CORBA 3.0, WML 등 인터넷 기반 프로그램과 제품에 사용

2. 자료의 표현 : 논리자료의 표현

❖ 논리자료

- 논리값을 표현하기 위한 자료 형식
- 논리값
  - 참(True)와 거짓(False), 1과 0
- 1바이트를 사용하여 논리자료를 표현하는 방법
  - 방법 1)
    - 참 : 최하위 비트를 1로 표시 **00000001**
    - 거짓 : 전체 비트를 0으로 표시. **00000000**
  - 방법2)
    - 참 : 전체 비트를 1로 표시. **11111111**
    - 거짓 : 전체 비트를 0으로 표시. **00000000**
  - 방법3)
    - 참 : 하나 이상의 비트를 1로 표시 **00000001** *or* **00000100 .....**
    - 거짓 : 전체 비트를 0으로 표시. **00000000**

2. 자료의 표현 : 포인터 자료의 표현

❖ 포인터 자료

- 메모리의 주소를 표현하기 위한 자료 형식
- 변수의 주소나 메모리의 특정 위치에 대한 주소를 저장하고 주소연산하기 위해 사용

2. 자료의 표현 : 문자열 자료의 표현

❖ 문자열String 자료

- 여러 문자로 이루어진 문자의 그룹을 하나의 자료로 취급하여 메모리에 연속적으로 저장하는 자료 형식

- 하나의 문자열 자료에 포함된 부분문자열을 표현하는 방법
  - 방법 1 : 부분 문자열 사이에 구분자를 사용하여 저장한다.
  - 방법 2 : 가장 긴 문자열의 길이에 맞춰 고정 길이로 저장한다.
  - 방법 3 : 부분 문자열을 연속하여 저장하고 각 부분 문자열에 대한 포인터를 사용한다.

2. 자료의 표현 : 문자열 자료의 표현

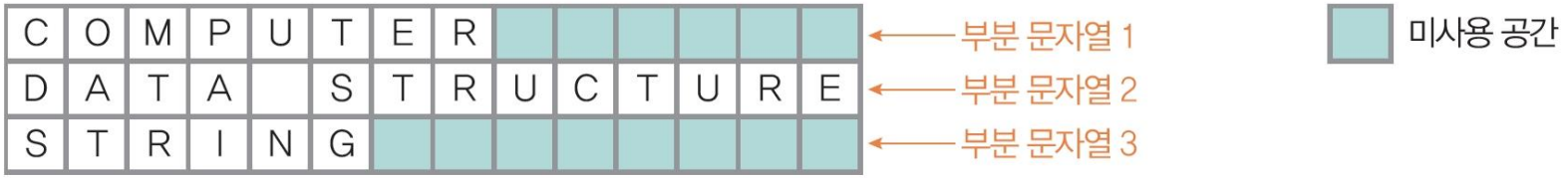
❖ 문자열String 자료

- 방법 1. 구분자를 사용하는 표현 : 구분자로 세미콜론(;) 사용

(a)

- 방법 2. 고정길이를 사용하는 표현

(b)



- 방법 3. 포인터를 사용하는 표현

(c)

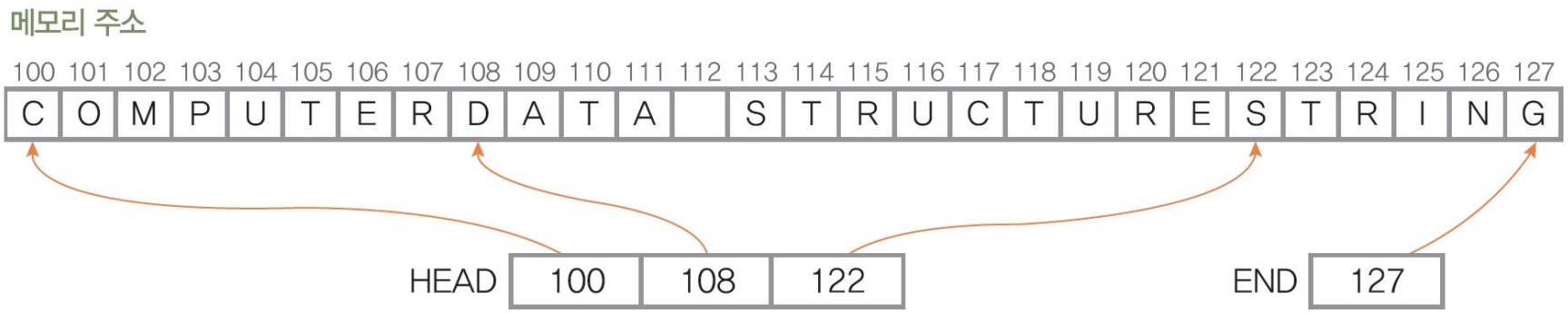


그림 1-22 문자열 자료의 표현 예 : {COMPUTER, DATA STRUCTURE, STRING} 저장

2. 자료의 표현 : 문자열 자료의 표현

❖ 문자열String 자료

표 1-7 문자열 표현 방법 비교

비교 항목 / 방법	메모리 이용률	부분 문자열 탐색 시간
구분자를 사용하는 방법	문자열 길이 + 구분자 길이 → 효율적	문자 비교 연산 시간 + 구분자 식별 시간 → 비효율적
고정 길이로 저장하는 방법	가장 긴 부분 문자열 길이 × 부분 문자열의 개수 → 비효율적	문자 비교 연산 시간 → 효율적
포인터를 사용하는 방법	문자열 길이 + 포인터 저장 공간 → 효율적	문자 비교 연산 시간 + 포인터 주소 연산 시간 → 효율적

3. 자료의 추상화

❖ 뇌의 추상화 기능

- 기억할 대상의 구별되는 특징만을 단순화하여 기억하는 기능

그림 1-23 뇌의 추상화 기능

3. 자료의 추상화

❖ 컴퓨터를 이용한 문재해결에서의 추상화

- 크고 복잡한 문제를 단순화시켜 쉽게 해결하기 위한 방법
- 자료 추상화(Data Abstraction)
  - 처리할 자료, 연산, 자료형에 대한 추상화 표현
  - 자료 : 프로그램의 처리 대상이 되는 모든 것을 의미
  - 연산
    - 어떤 일을 처리하는 과정. 연산자에 의해 수행
    - 예) 더하기 연산은 +연산자에 의해 수행
  - 자료형
    - 처리할 자료의 집합과 자료에 대해 수행할 연산자의 집합
    - 예) 정수 자료형

      자료 : 정수의 집합. {…, -1, 0, 1, …}

      연산자 : 정수에 대한 연산자 집합. {+, -, x, ÷, mod}

3. 자료의 추상화 : 개념

❖ 추상 자료형(ADT, Abstract Data Type)

- 자료와 연산자의 특성을 논리적으로 추상화하여 정의한 자료형

❖ 추상화와 구체화

- 추상화 – "무엇(what)인가?"를 논리적으로 정의
- 구체화 – "어떻게(how) 할 것인가?"를 실제적으로 표현

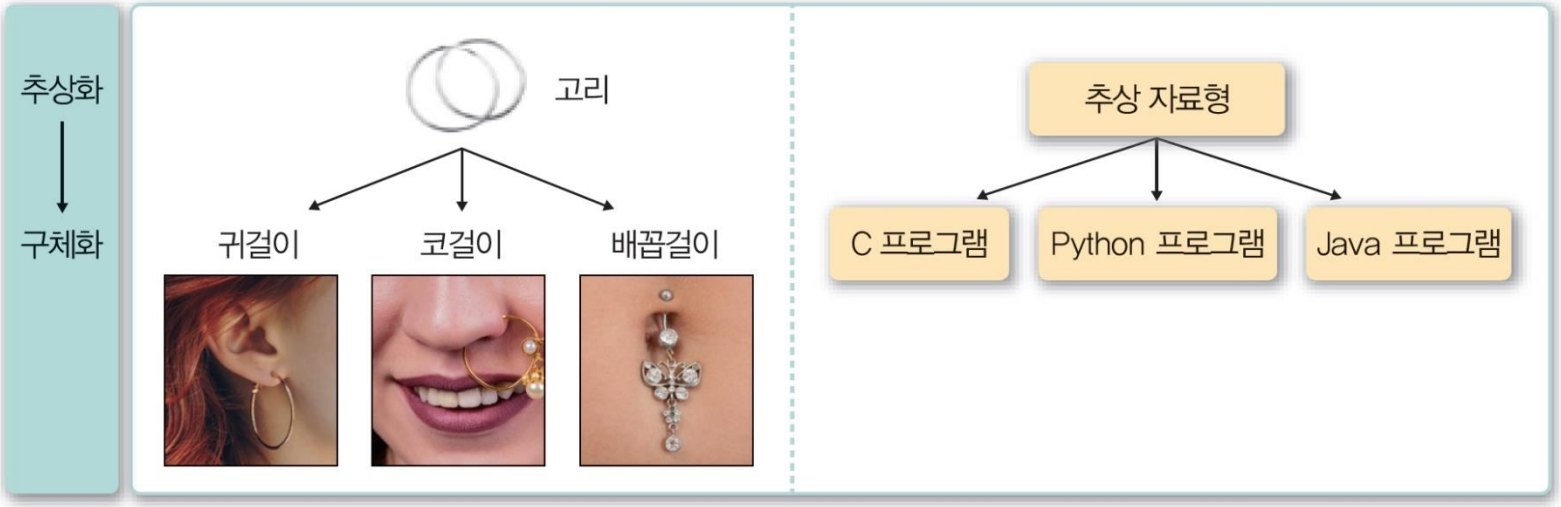


그림 1-25 추상화와 구체화의 예

3. 자료의 추상화

❖ 추상화와 구체화

- 자료와 연산에 있어서의 추상화와 구체화의 관계

표 1-8 자료와 연산의 추상화와 구체화

구분	자료	연산
추상화	추상 자료형	알고리즘 정의
구체화	자료형	프로그램 구현

4. 알고리즘의 이해

❖ 알고리즘

- 문제해결 방법을 추상화하여 단계적 절차를 논리적으로 기술해 놓은 명세서

https://www.youtube.com/watch?v=l5cq54MFQCo

❖ 알고리즘의 조건

- 입력input : 알고리즘 수행에 필요한 자료가 외부에서 입력으로 제공될 수 있어야 한다.
- 출력output : 알고리즘 수행 후 하나 이상의 결과를 출력해야 한다.
- 명확성definiteness : 수행할 작업의 내용과 순서를 나타내는 알고리즘의 명령어들은 명확하게 명세되어야 한다.
- 유한성finiteness : 알고리즘은 수행 뒤에 반드시 종료되어야 한다.
- 효과성effectiveness : 알고리즘의 모든 명령어들은 기본적이며 실행이 가능해야 한다.

4. 알고리즘의 이해

자료

요리 재료

케이크 시트(20cm×20cm) 1개, 크림치즈 무스(크림치즈 200g, 달걀 2알, 설탕 3큰술, 레몬즙 1큰술, 바닐라 에센스 1큰술), 딸기 시럽(딸기 500g, 설탕 1½컵, 레몬즙 1작은술), 딸기 1개, 플레인 요구르트 2큰술

절차

요리법 **알고리즘**

1. 케이크 틀에 유산지를 깔고 케이크 시트를 놓는다.
2. 달걀 2알을 잘 푼다. 볼에 크림치즈를 넣고 거품기로 젓는다. 달걀 푼 물과 설탕 3큰술을 세 차례로 나누어 넣으면서 크림 상태가 되도록 거품기로 젓는다.
3. ❷에 레몬즙과 바닐라 에센스를 넣고 살짝 저은 다음 ❶에 붓는다. 180℃로 예열된 오븐에 전체를 넣고 20분 정도 굽는다.
4. 딸기를 얇게 자르고 냄비에 넣은 다음 설탕 1½컵을 넣고 약한 불로 끓인다. 눌어붙지 않도록 계속해서 젓고 거품이 생기면 걷어 낸다. 되직해지면 레몬즙을 넣고 차갑게 식힌다.
5. 치즈케이크 한 조각을 접시에 담고 ❹를 뿌린 다음 플레인 요구르트와 딸기를 얹는다.

연산

그림 1-27 딸기 시럽을 얹은 치즈케이크 만들기

5. 알고리즘의 표현 방법

❖ 알고리즘의 표현 방법의 종류

- 자연어를 이용한 서술적 표현 방법
- 순서도Flow chart를 이용한 도식화 표현 방법
- 프로그래밍 언어를 이용한 구체화 방법
- 가상코드Pseudo-code를 이용한 추상화 방법

5. 알고리즘의 표현 방법

❖ 순서도를 이용한 도식화

- 순서도의 예) 1부터 5까지의 합을 구하는 알고리즘

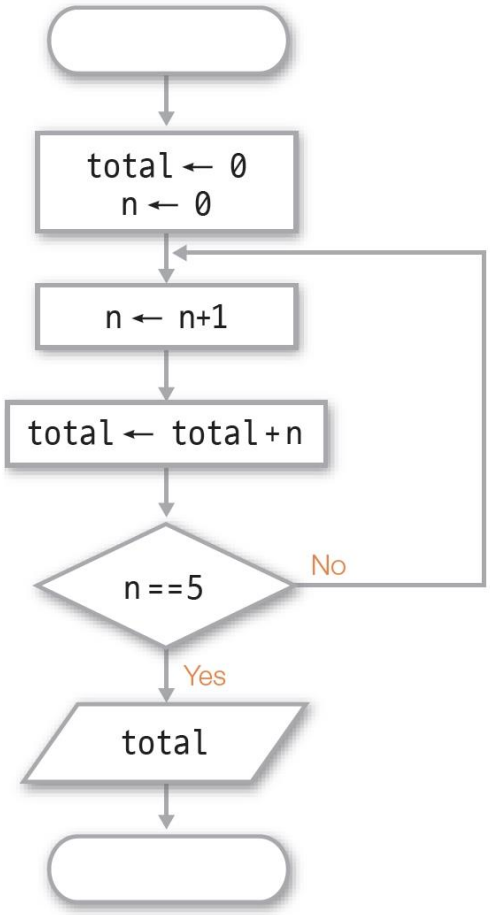
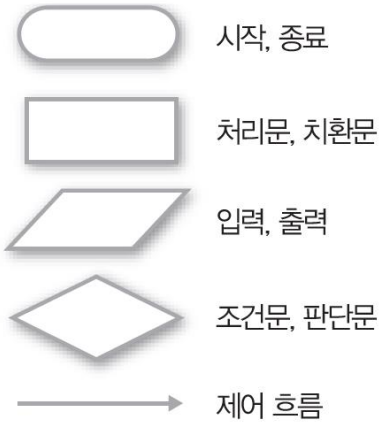


그림 1-28 순서도의 예



5. 알고리즘의 표현 방법

❖ 가상코드를 이용한 추상화

- 가상코드, 즉 알고리즘 기술언어ADL, Algorithm Description Language를 사용하여 프로그래밍 언어의 일반적인 형태와 유사하게 알고리즘을 표현
- 특정 프로그래밍 언어가 아니므로 직접 실행은 불가능
- 일반적인 프로그래밍 언어의 형태이므로 원하는 특정 프로그래밍 언어로의 변환 용이

5. 알고리즘의 표현 방법

❖ 가상코드의 형식

- 기본 요소
  - 기호
    - 변수, 자료형 이름, 프로그램 이름, 레코드 필드 명, 문장의 레이블 등을 나타냄
    - 문자나 숫자의 조합. 첫 문자는 반드시 영문자 사용.
  - 자료형
    - 정수형과 실수형의 수치 자료형, 문자형, 논리형, 포인터, 문자열 등의 모든 자료형 사용
  - 연산자
    - 산술연산자, 관계연산자, 논리연산자

- 지정문 형식과 예

```
변수 ← 값;
```

(a) 지정문 형식

```
a ← 5;
a ← 3+2;
a ← b;
```

(b) 지정문 예

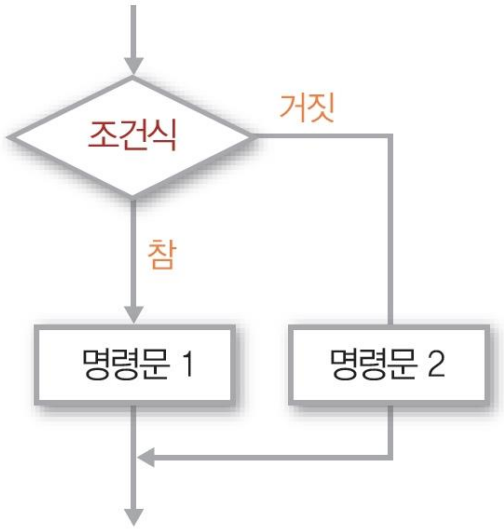
그림 1-29 지정문 형식과 예

5. 알고리즘의 표현 방식

❖ 조건문

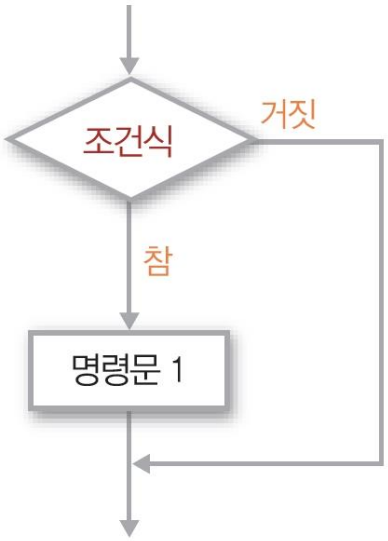
- 조건에 따라 실행할 명령문이 결정되는 선택적 제어구조를 만든다.
- if 문의 형식과 제어흐름

```
if (조건식) then 명령문 1;
else 명령문 2;
```



(a) if – then – else 형

```
if (조건식) then 명령문 1;
```



(b) if – then 형

그림 1-30 기본 if 문의 형식과 제어 흐름

5. 알고리즘의 표현 방식

- 다단계 조건문
  - 중첩 if 문의 형식과 제어 흐름

```
if (조건식 1) then 명령문 1;
else if (조건식 2) then 명령문 2;
else 명령문 3;
```

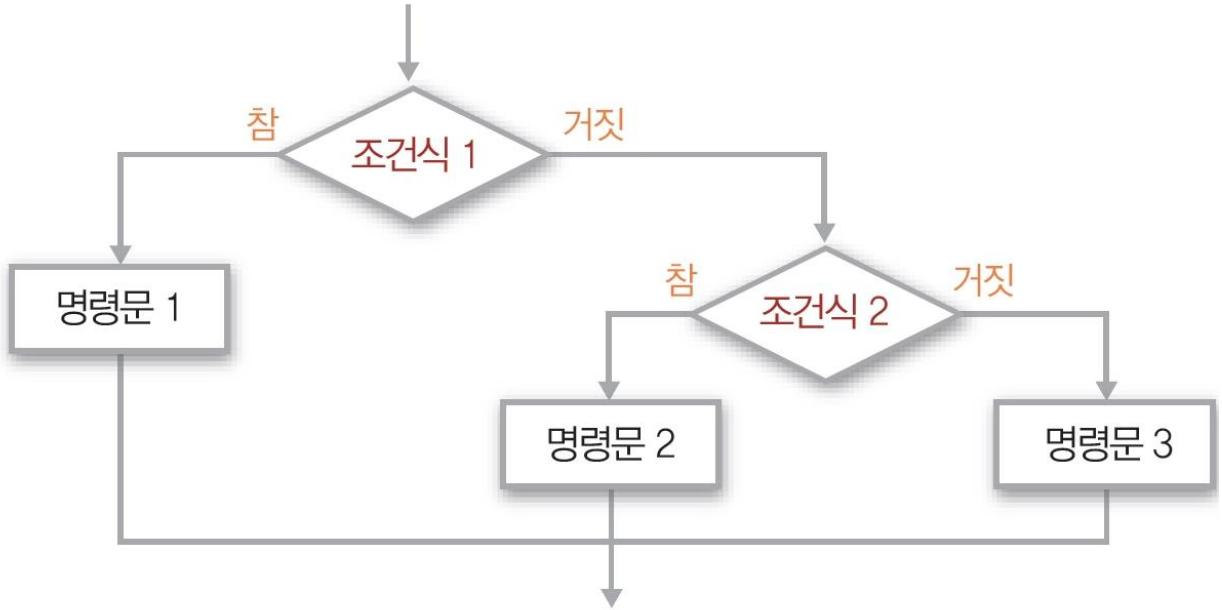


그림 1-31 중첩 if 문의 형식과 제어 흐름

5. 알고리즘의 표현 방식

- 중첩 if문 사용 예) 평균 점수에 따른 등급 계산하기

```
if Average >= 90 then grade ← 'A';
else if Average >= 80 then grade ← 'B';
else if Average >= 70 then grade ← 'C';
else grade ← 'F';
```

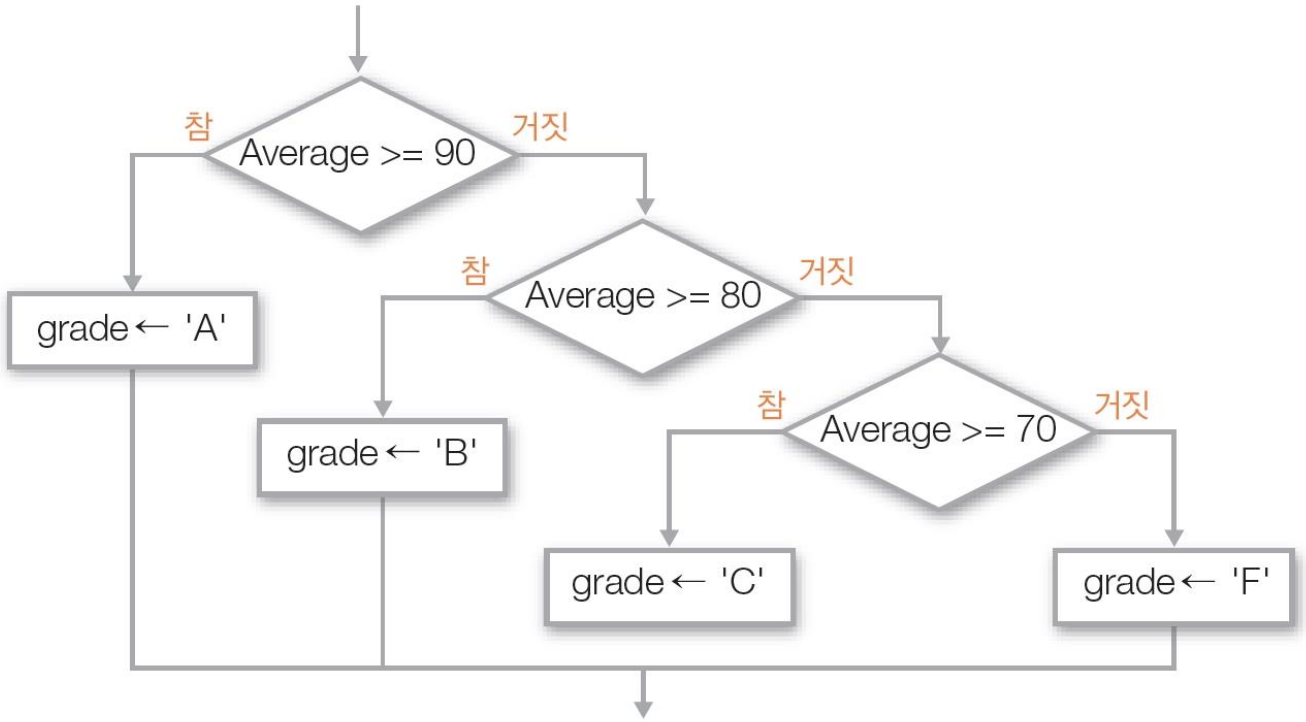


그림 1-32 중첩 if 문의 예

5. 알고리즘의 표현 방식

- case 문
  - 여러 조건식 중에서 해당 조건을 찾아서 그에 대한 명령문을 수행
  - 중첩 if 문으로 표현 가능
  - 형식과 제어흐름

```
case {
    조건식 1 : 명령문 1;
    조건식 2 : 명령문 2;
    ...
    조건식 n : 명령문 n;
    else    : 명령문 n+1;
}
```

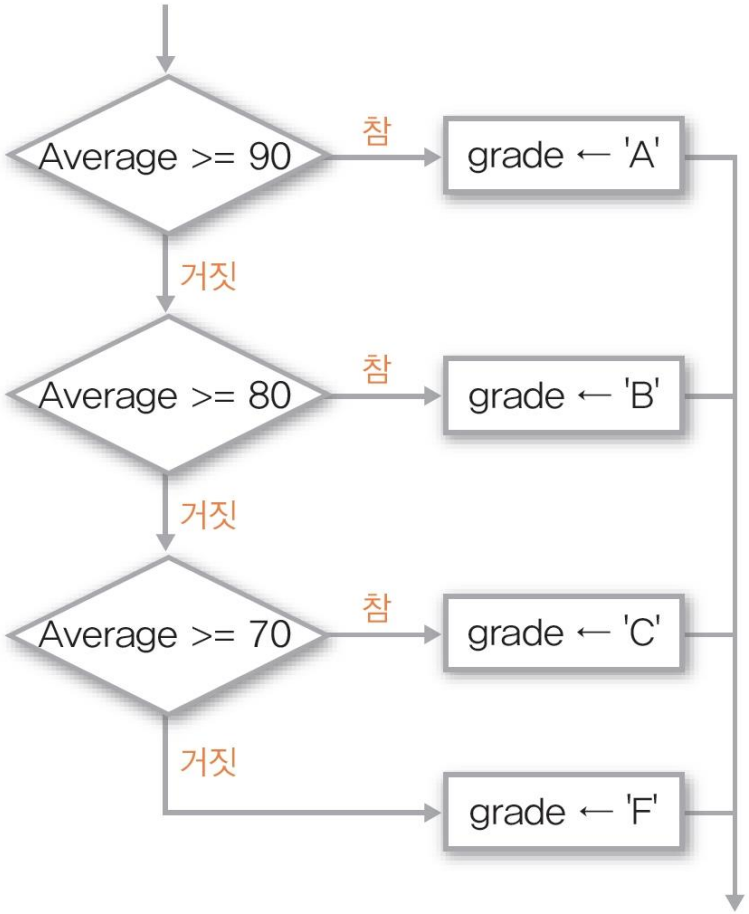
그림 1-33 case 문의 형식과 제어 흐름

5. 알고리즘의 표현 방식

- case 문
  - 예) 평균 점수에 따른 등급 계산하기

```
case {
    Average >= 90 : grade ← 'A';
    Average >= 80 : grade ← 'B';
    Average >= 70 : grade ← 'C';
    else :          grade ← 'F';
}
```

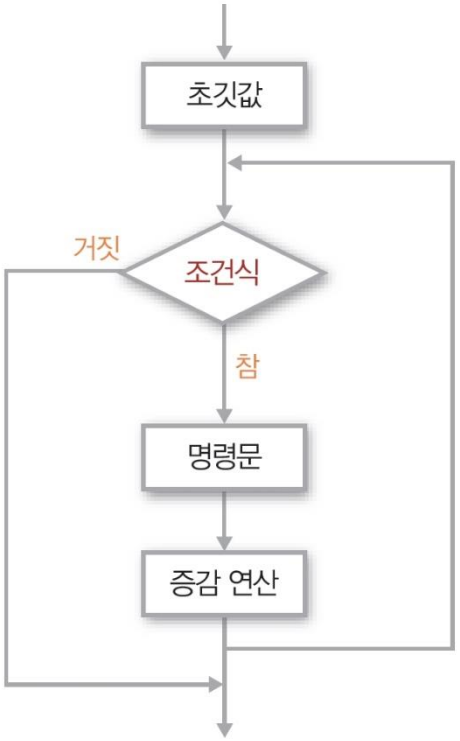
그림 1-34 case 문의 예



5. 알고리즘의 표현 방식

❖ 반복문

- 일정한 명령을 반복 수행하는 루프(loop) 형태의 제어구조
- for 문
  - 형식과 제어흐름




```
for (초깃값; 조건식; 증감값) do 명령문;
```

그림 1-35 for 문의 형식과 제어 흐름

5. 알고리즘의 표현 방식

- while – do 문
  - 형식과 제어흐름

```
while (조건식) do 명령문;
```

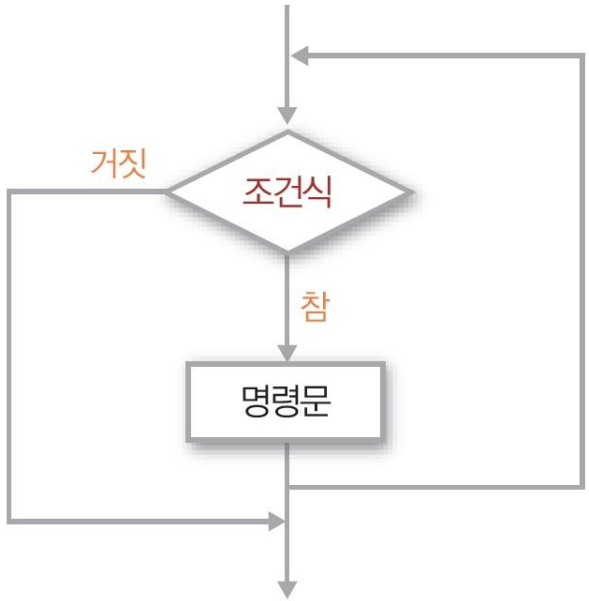


그림 1-36 while-do 문의 형식과 제어 흐름

5. 알고리즘의 표현 방식

- do-while 문
  - 형식과 제어흐름

```
do 명령문;
while (조건식);
```

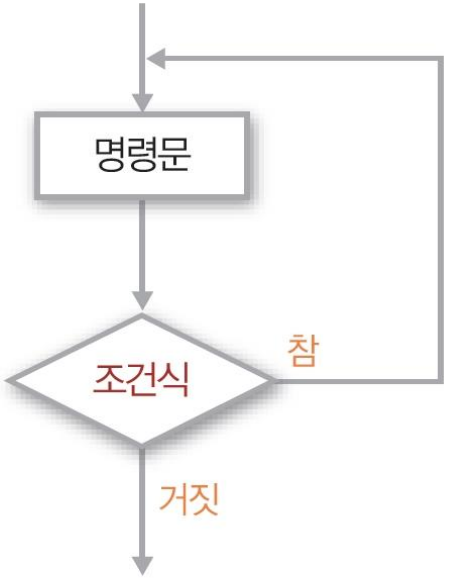


그림 1-37 do-while 문의 형식과 제어 흐름

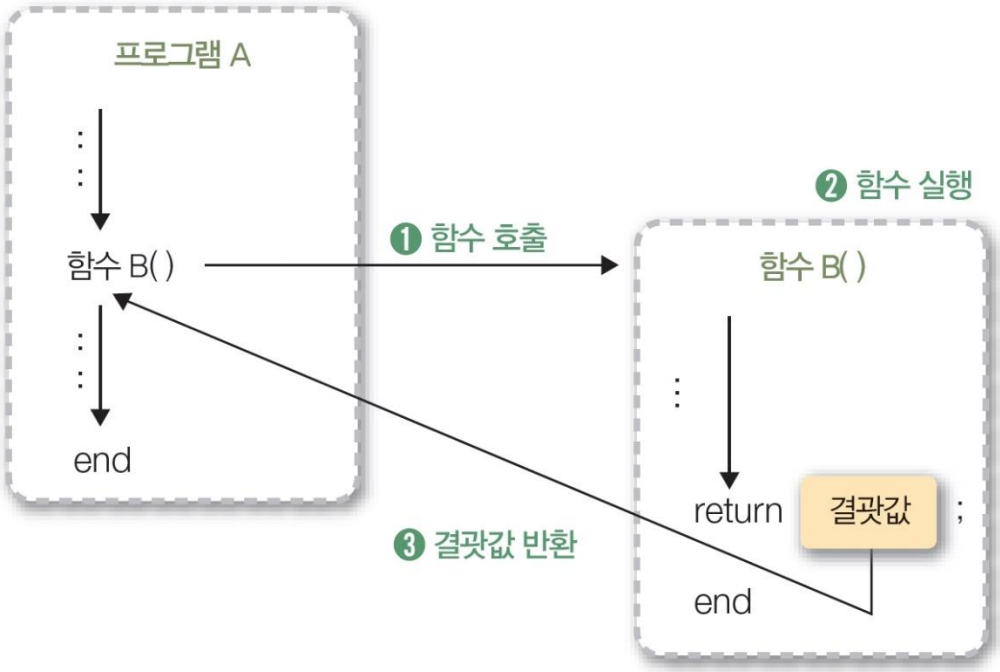
5. 알고리즘의 표현 방식

❖ 함수문

- 처리작업 별로 모듈화하여 만든 부프로그램
- 형식과 예

```
함수 이름(매개변수)
    명령문;
    ...
    return 결괏값;
end
```

(a) 함수의 형식



(b) 함수의 호출과 실행 및 결괏값 반환의 예

그림 1-38 함수의 형식과 예

6. 알고리즘의 성능분석

❖ 알고리즘 성능 분석 기준

- 기준에는 정확성, 명확성, 수행량, 메모리 사용량, 최적성 등 있음
  - 정확성 : 올바른 자료 입력 시 유한한 시간 내에 올바른 결과 출력 여부
  - 명확성 : 알고리즘이 얼마나 이해하기 쉽고 명확하게 작성되었는가
  - 수행량 : 일반적인 연산 제외, 알고리즘 특성 나타내는 중요 연산 모두 분석
  - 메모리 사용량
  - 최적성 : 가장 중요

6. 알고리즘의 성능 분석

❖ 알고리즘 성능 분석 방법

- 공간 복잡도
  - 알고리즘을 프로그램으로 실행하여 완료하기까지 필요한 총 저장 공간의 양
  - 공간 복잡도 = 고정 공간 + 가변 공간

- 시간 복잡도
  - 알고리즘을 프로그램으로 실행하여 완료하기까지의 총 소요시간
  - 시간 복잡도 = 컴파일 시간 + 실행 시간
    - 컴파일 시간 : 프로그램마다 거의 고정적인 시간 소요
    - 실행 시간 : 컴퓨터의 성능에 따라 달라질 수 있으므로 실제 실행시간 보다는 명령문의 실행 빈도수에 따라 계산
  - 실행 빈도수의 계산
    - 지정문, 조건문, 반복문 내의 제어문과 반환문은 실행시간 차이가 거의 없으므로 하나의 단위시간을 갖는 기본 명령문으로 취급

6. 알고리즘의 성능 분석

- 시간 복잡도 예)

알고리즘 1-1 피보나치 수열

```
00  fibonacci(n)
01      if (n<0) then
02          stop;
03      if (n≤1) then
04          return n;
05      f1 ← 0;
06      f2 ← 1;
07      for (i ← 2; i≤n; i ← i + 1) do {
08          fn ← f1 + f2;
09          f1 ← f2;
10          f2 ← fn;
11      }
12      return fn;
13  end
```

6. 알고리즘의 성능 분석

- 시간 복잡도 n<0, n=0, n=1의 경우에 대한 실행 빈도수

표 1-9 피보나치 수열 알고리즘의 실행 빈도수

(a) n〈0, n=0, n=1인 경우

행 번호	n〈0	n=0	n=1
01	1	1	1
02	1	0	0
03	0	1	1
04	0	1	1
05~13	0	0	0

(b) n〉1인 경우

행 번호	n〉1	행 번호	n〉1
01	1	08	n-1
02	0	09	n-1
03	1	10	n-1
04	0	11	0
05	1	12	1
06	1	13	0
07	n		

6. 알고리즘의 성능 분석

❖ 알고리즘 성능 분석 표기법

- 빅-오 표기법
  - O(f(n))과 같이 표기, "Big Oh of f (n)"으로 읽음
  - 수학적 정의
    - 함수 f(n)과 g(n)이 주어졌을 때, 모든 n≥n0에 대하여 |f(n)| ≤ c |g(n)|을 만족하는 상수 c와 n0이 존재하면, f(n) = O(g(n))이다.
  - 함수의 상한을 나타내기 위한 표기법
    - 최악의 경우에도 g(n)의 수행 시간 안에는 알고리즘 수행 완료 보장
  - 먼저 실행 빈도수를 구하여 실행 시간 함수를 찾고, 이 함수값에 가장 큰 영향을 주는 n에 대한 항을 한 개 선택하여 계수는 생략하고 O의 오른쪽 괄호 안에 표시
  - [알고리즘 1-1]의 피보나치 수열에서 실행 시간 함수는 4n+2이고, 가장 영향이 큰 항은 4n인데 여기에서 계수 4를 생략하여 O(n)으로 표기

6. 알고리즘의 성능 분석

- 빅-오메가 표기법
  - Ω(f(n))과 같이 표기, "Big Omega of f (n)"으로 읽음
  - 수학적 정의
    - 함수 f(n)과 g(n)이 주어졌을 때, 모든 n≥n0에 대하여 |f(n)| ≥ c |g(n)|을 만족하는 상수 c와 n0이 존재하면, f(n) = Ω(g(n))이다.
  - 함수의 하한을 나타내기 위한 표기법
  - 어떤 알고리즘의 시간 복잡도가 Ω(g(n))으로 분석되었다면, 이 알고리즘 수행에는 적어도 g(n)의 수행 시간이 필요함을 의미

6. 알고리즘의 성능 분석

- 빅-세타 표기법
  - θ(f(n))과 같이 표기, "Big Theta of f (n)"으로 읽음
  - 수학적 정의
    - f(n)과 g(n)이 주어졌을 때, 모든 $n \geq n_0$에 대하여 $c|g(n)| \leq f(n) \leq c|g(n)|$을 만족하는 상수 c_1, c_2와 n_0이 존재하면, $f(n) = \theta(g(n))$이다
  - 상한과 하한이 같은 정확한 차수를 표현하기 위한 표기법
    - $f(n) = \theta(g(n))$이 되려면 $f(n) = O(g(n))$이면서 $f(n) = \Omega(g(n))$이어야 함

6. 알고리즘의 성능 분석

- 각 실행 시간 함수에서 n값의 변화에 따른 실행 빈도수 비교

logn	<	n	<	nlogn	<	n^2	<	n^3	<	2^n
0		1		0		1		1		2
1		2		2		4		8		4
2		4		8		16		64		16
3		8		24		64		512		256
4		16		64		256		4096		65536
5		32		160		1024		32768		4294967296

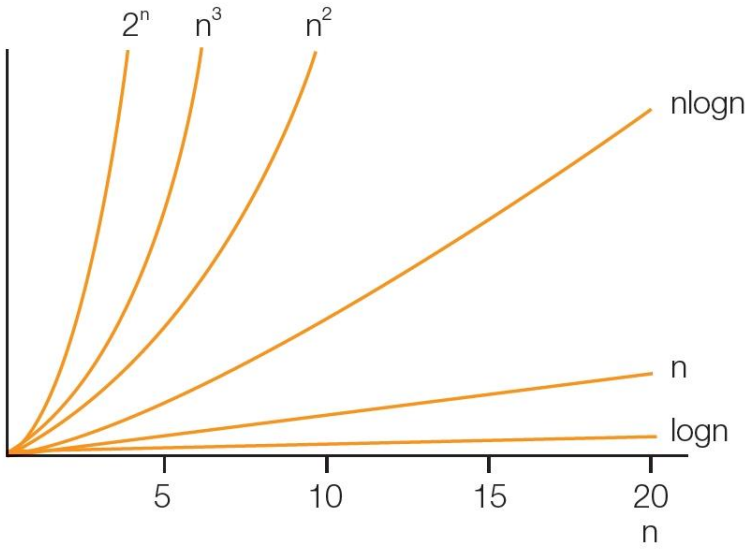


그림 1-39 실행 시간 함수에서 n값의 변화에 따른 실행 빈도수 비교

6. 알고리즘의 성능 분석

- 시간 복잡도에 따른 알고리즘 수행 시간 비교

표 1-10 시간 복잡도에 따른 알고리즘 수행 시간 비교 예

입력 크기 n	알고리즘 수행 시간				
	n	nlogn	n^2	n^3	2^n
10	10^{-8}초	3×10^{-8}초	10^{-7}초	10^{-6}초	10^{-6}초
30	3×10^{-8}초	2×10^{-7}초	9×10^{-7}초	3×10^{-5}초	1초
50	5×10^{-8}초	3×10^{-7}초	3×10^{-6}초	10^{-4}초	13일
100	10^{-7}초	7×10^{-7}초	10^{-5}초	10^{-3}초	4×10^{13}년
1,000	10^{-6}초	10^{-5}초	10^{-3}초	1초	3×10^{283}년
10,000	10^{-5}초	10^{-4}초	10^{-1}초	17분	
100,000	10^{-4}초	2×10^{-3}초	10초	12일	
1,000,000	10^{-3}초	2×10^{-2}초	17분	32년	