



PART 02. C 언어 기초 문법

CHAPTER 03. 변수와 자료형

Section 3.1

변수

1. 변수

■ 변수의 개념

- 컴퓨터 프로그램이 실행되는 동안 데이터를 저장하고 그 데이터를 참조하거나 수정하기 위해 사용하는 식별자
- 변수를 선언하면 메모리에 공간이 할당되고 해당 메모리 위치에 데이터 저장

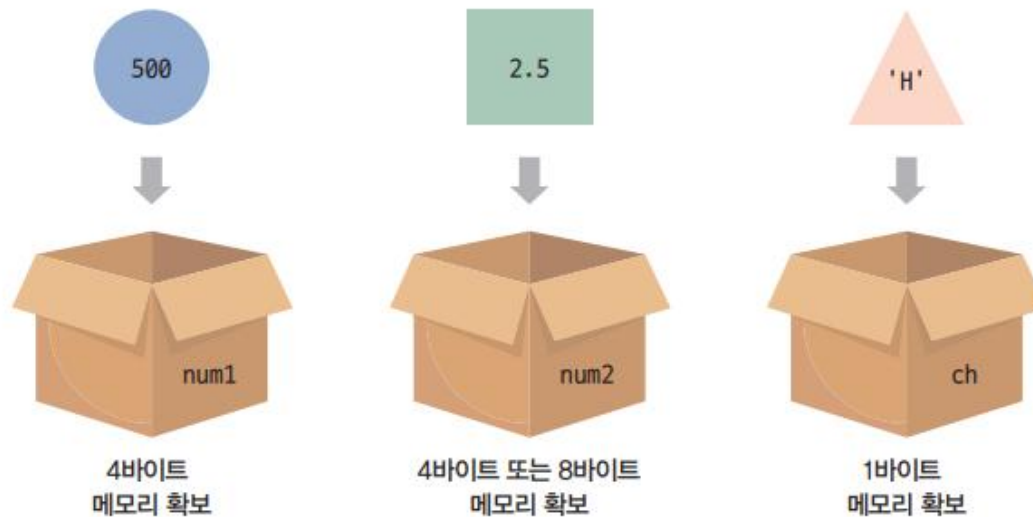


그림 3-1 선언된 변수와 데이터 크기에 따른 메모리 공간

1. 변수

■ 변수의 개념

- 선언된 변수는 자료형의 크기만큼 RAM에 공간을 할당받고, 해당 변수의 값 저장

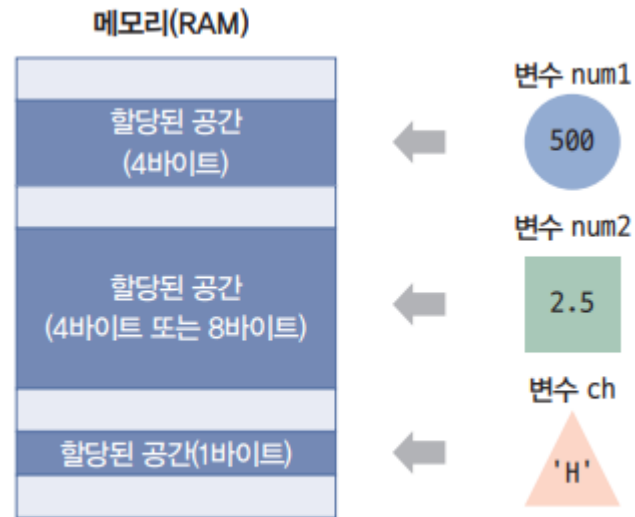


그림 3-2 메모리 공간에 할당된 변수 num1, num2, ch

1. 변수

■ 변수의 사용

■ 변수 선언

- 변수의 이름과 자료형을 지정하여 변수를 생성하는 과정

| Syntax | 변수 선언

자료형 변수명;
`int var;`

```
int myVariable;    // 정수형 변수 myVariable 선언  
myVariable = 17;   // 변수에 정수 17 대입
```

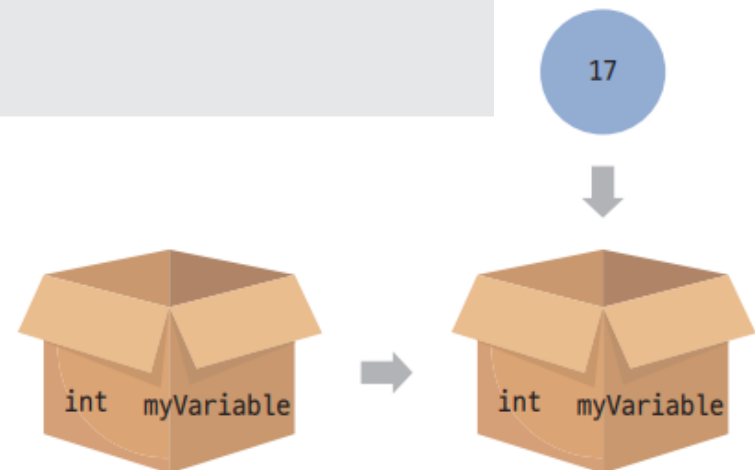


그림 3-3 변수 선언과 값 대입의 개념

1. 변수

■ 변수의 사용

■ 변수 초기화

- 생성한 변수에 초기값을 할당하는 과정

| Syntax | 변수 초기화

자료형 변수명 = 값;

`int xNum = 0;`

4바이트 크기, int형 변수값을 0으로 초기화

`double yNum = 3.3;`

8바이트 크기, double형 변수값을 3.3으로 초기화

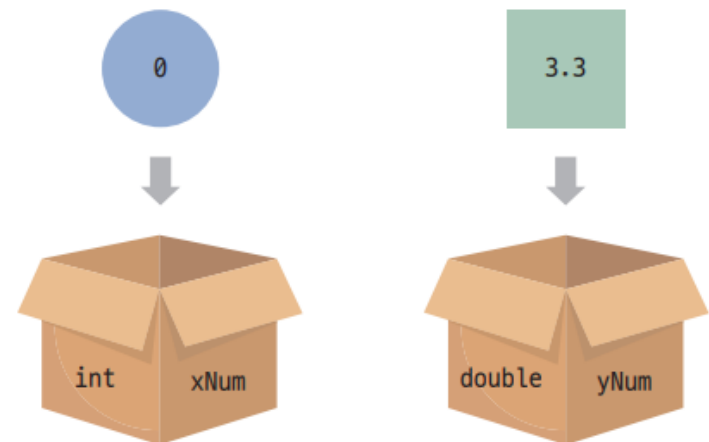


그림 3-4 변수 초기화

1. 변수

■ 변수의 사용

■ 변수명 작성

- 영문자(대·소문자), 숫자, 언더바(_) 사용 가능
- 변수명의 첫 글자는 숫자로 시작 불가능
- 대문자와 소문자 구분
- 변수명 중간에 공백 사용 불가능
- C 언어의 예약어 사용 불가능

```
countValue, student_code, _personName, DATETIME, ... // 올바른 변수명
15_value, #Test, sum value, ...                      // 잘못된 변수명
Master, master, MASTER, mAsTer                      // 각기 다른 변수
```

1. 변수

■ 변수의 사용

■ 변수를 사용하는 이유

- 값을 저장하는 메모리 공간으로 사용
- 데이터 유지와 재사용 가능
- 효율적인 계산과 연산 가능

변수를 사용한 간단한 연산

variable.c

```
01 #include <stdio.h>
02
03 int main()
04 {
05     int a, b, tot;
06
07     a = 200;
08     b = 100;
09     tot = a + b;
10
11     printf("%d", tot); // tot 출력
12
13     return 0;
14 }
```

정수형 변수 a, b, tot 선언

a의 초기값에 200 대입

b의 초기값에 100 대입

300

1. 변수

■ 변수의 사용

■ 변수를 사용하는 이유

- 프로그램 상태 표현에 사용

```
int pacman_hp = 25000;    // 팩맨의 생명력값 저장  
float ghost_locX = 120.2f; // 유령의 x 좌표값 저장  
float ghost_locY = 150.5f; // 유령의 y 좌표값 저장
```

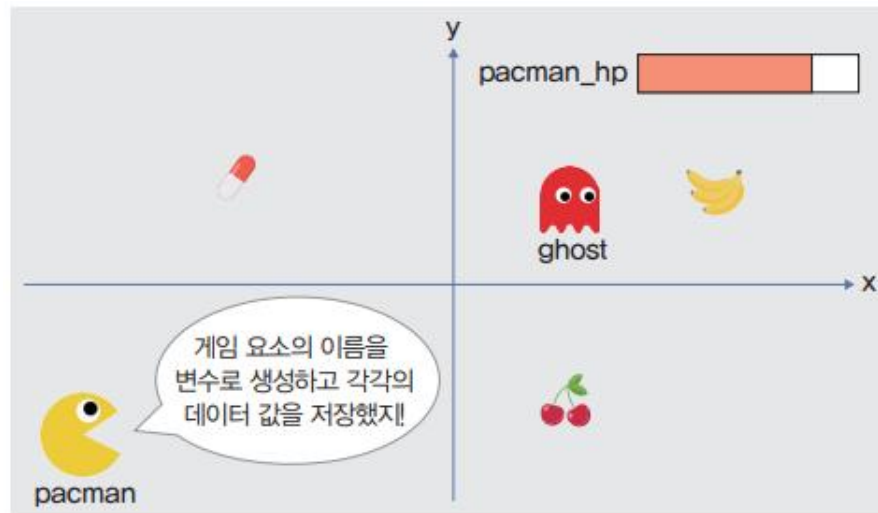


그림 3-5 변수를 활용한 게임 프로그램의 상태 표현

Section 3.2

자료형

2. 자료형

■ 자료형의 개념

- 변수에 데이터를 저장할 때 데이터의 종류와 크기를 정의하는 단위

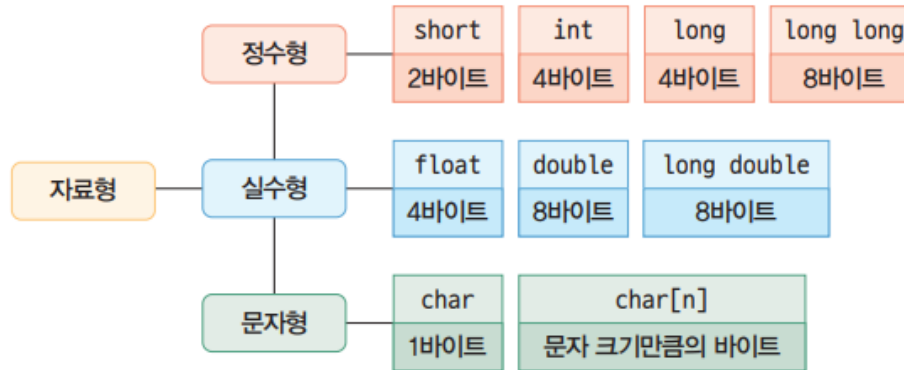


그림 3-6 자료형별 바이트 단위



그림 3-7 데이터의 유형과 크기에 맞는 자료형 사용

2. 자료형

■ 자료형의 종류

표 3-1 C 언어에서 제공하는 자료형의 종류

자료형		설명
정수형	short	int보다 작은 범위의 정수를 표현하는 데 사용된다.
	int	정수를 표현하는 데 사용되는 가장 일반적인 정수형이다.
	long	int보다 큰 범위의 정수를 표현하는 데 사용된다.
	long long	long보다 큰 범위의 정수를 표현하는 데 사용된다. 매우 큰 수를 다루거나 정밀한 연산을 할 때 유용하다.
실수형	float	단정밀도 부동 소수점 수를 표현하는 데 사용된다.
	double	배정밀도 부동 소수점 수를 표현하는 데 사용된다.
	long double	배정밀도 부동 소수점 수를 표현하는 데 사용된다. C 표준에서는 double의 최소 크기를 정의하고 있으며, long double은 그 이상의 크기와 정밀도를 나타낸다.
문자형	char	문자 1개를 표현하는 데 사용된다.
	char[n]	문자 n개를 표현하는 데 사용된다.

2. 자료형

하나 더 알기 변수 선언 시 자료형 범위의 오류

변수를 선언할 때 자료형과 저장되는 값의 범위가 맞지 않으면 메모리 공간을 할당할 때 오류가 발생한다. 하지만 비주얼 스튜디오 2022와 같은 에디터에서는 자료형 범위에 벗어난 값을 정의하면 빨간색 밑줄과 함께 오류 내용이 나타난다.

```
// 값보다 큰 범위 선언 시 오류 발생  
int a = 1657820170803929857060496847329
```

정수 상수가 너무 큼니다.

[온라인 검색](#)

그림 3-8 비주얼 스튜디오 2022의 오류 표시

2. 자료형

■ 정수형

■ 정수형 종류와 유효 범위

- 정수형: 정수(양수, 0, 음수)를 표현하기 위한 자료형

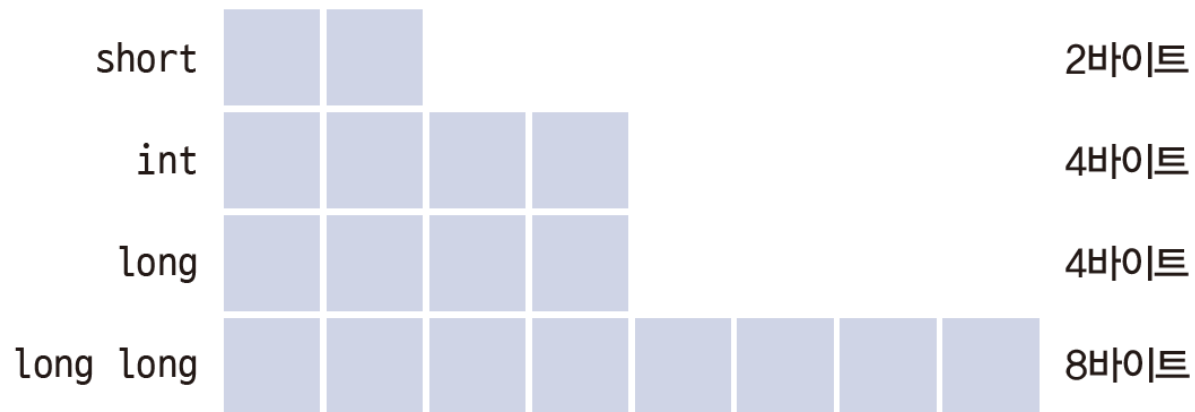


그림 3-9 정수형의 종류와 메모리 저장 공간

2. 자료형

■ 정수형

■ 정수형 종류와 유효 범위

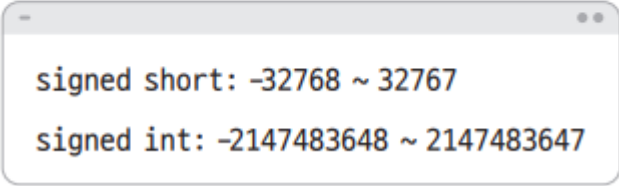
표 3-2 정수형의 종류와 유효 범위

정수형		크기	유효 범위
short	signed short	2바이트	-32,768 ~ 32,767
	unsigned short		0 ~ 65,535
int	signed int	4바이트	-2,147,483,648 ~ 2,147,483,647
	unsigned int		0 ~ 4,294,967,295
long	signed long	4바이트	-2,147,483,648 ~ 2,147,483,647
	unsigned long		0 ~ 4,294,967,295
long long	signed long long	8바이트	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
	unsigned long long		0 ~ 18,446,744,073,709,551,615

2. 자료형

예제 3-1 정수형 변수를 선언하고 값 출력하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     signed short smallest_short = -32768;
06     signed short largest_short = 32767;
07
08     signed int smallest_int = -2147483648;
09     signed int largest_int = 2147483647;
10
11     printf("signed short: %d ~ %d\n", smallest_short, largest_short);
12     printf("signed int: %d ~ %d\n", smallest_int, largest_int);
13
14     return 0;
15
16 }
```



```
signed short: -32768 ~ 32767
signed int: -2147483648 ~ 2147483647
```


2. 자료형

■ 정수형

▪ sizeof 연산자

- 주어진 변수나 자료형의 크기를 바이트 단위로 알림
- sizeof 연산자로 확인한 크기는 컴파일 시 확정

| Syntax | sizeof 연산자

sizeof (피연산자); 자료형, 변수, 상수도 사용 가능

sizeof 변수명; 자료형 사용 불가

2. 자료형

예제 3-2 sizeof 연산자를 사용하여 정수형의 크기 출력하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     short sh = 12;        // short 정수형 변수
06     int nt = 155;         // int 정수형 변수
07     long long on = 1666;  // long long 정수형 변수
08
09     printf("자료형의 크기를 알아보는 코드 \n");
10     printf("1. short : %dbyte, %dbyte \n", sizeof(sh), sizeof sh);
11     printf("2. int : %dbyte, %dbyte \n", sizeof(nt), sizeof nt);
12     printf("3. long long : %dbyte, %dbyte \n", sizeof(on), sizeof on);
13
14     return 0;
15 }
```

자료형의 크기를 알아보는 코드

```
1. short : 2byte, 2byte
2. int : 4byte, 4byte
3. long long : 8byte, 8byte
```

2. 자료형

■ 정수형

▪ 정수형의 부호

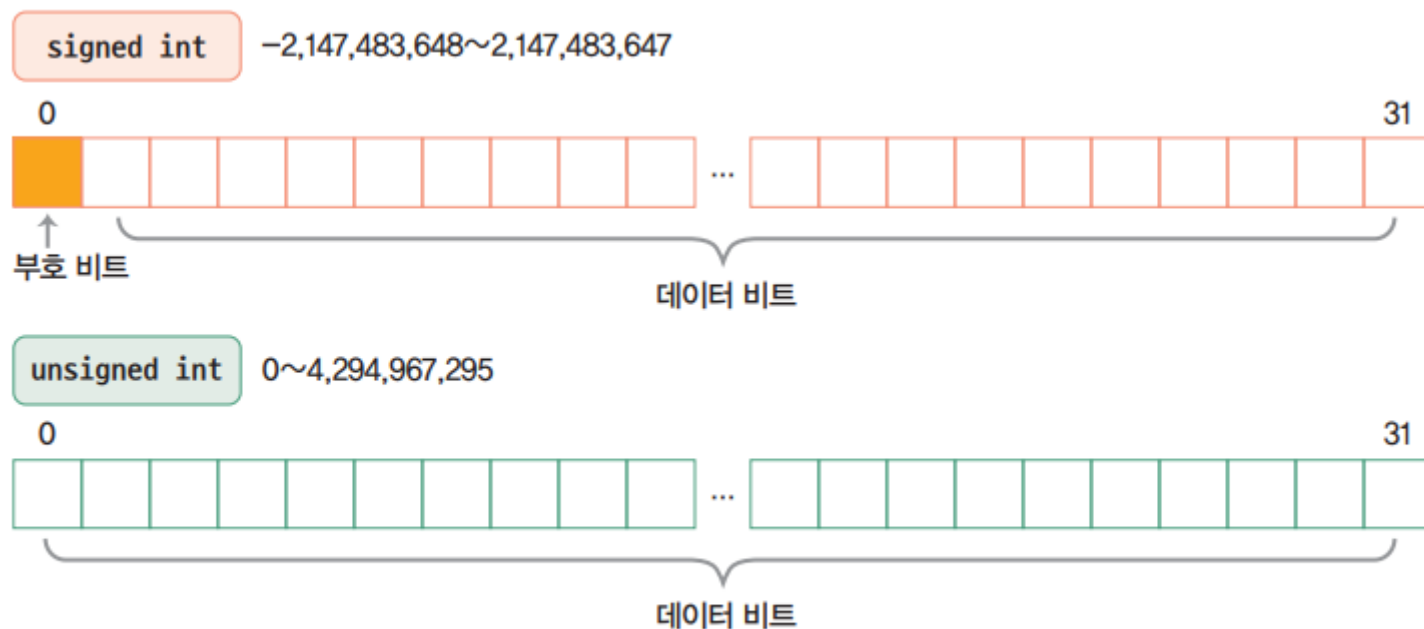


그림 3-11 부호 표시 여부에 따라 달라지는 int형

2. 자료형

■ 정수형

▪ 부호 있는 정수 signed

- 부호 있는 정수를 나타내며 양수, 0, 음수를 포함하는 모든 정숫값 저장 가능

$-2^{31}, \dots, -1, 0, 1, \dots, 2^{31}-1$
 $-2,147,483,648 \sim +2,147,483,647$

▪ 부호 없는 정수 unsigned

- 부호 없는 정수를 나타내며 0과 양수만 저장 가능

$0, 1, 2, \dots, 2^{32}-1$
 $0 \sim +4,294,967,295$

```
signed int myNumber = -1024;      // 부호 있는 정수
unsigned int positiveNumber = 1024; // 부호 없는 정수
```

2. 자료형

■ 정수형

■ 오버플로

- 선언된 변수 자료형의 범위를 초과하는 값을 저장할 때 발생하는 현상



그림 3-12 오버플로의 개념

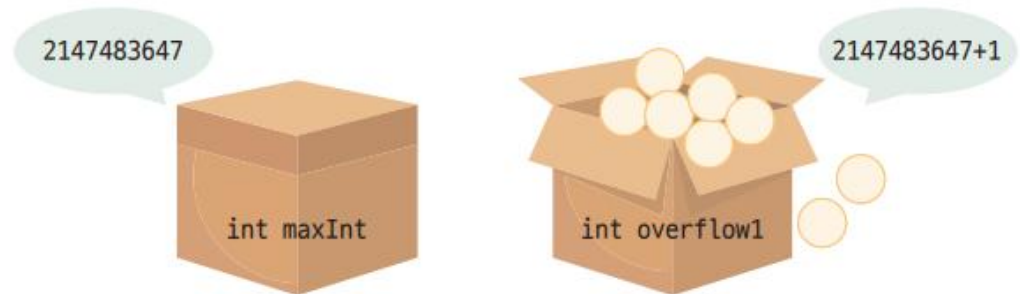


그림 3-13 변수의 최댓값을 초과하는 오버플로

■ 언더플로

- 변수로 선언한 자료형의 유효 범위보다 더 작은 값이 대입될 때 발생하는 현상

2. 자료형

예제 3-3 int 자료형에서 오버플로가 발생하는 경우

```
01 #include <stdio.h>
02
03 int main()
04 {
05     int maxInt = 2147483647;    // int의 최댓값으로 초기화
06     int overflow1 = maxInt + 1;
07                                     오버플로가 발생하는 연산
08     printf("maxInt: %d\n", maxInt);
09     printf("overflow1: %d\n", overflow1);
10
11     unsigned int maxUInt = 4294967295U;    // unsigned int의 최댓값
12     unsigned int overflow2 = maxUInt + 1;
13                                     오버플로가 발생하는 연산
14     printf("maxUInt: %u\n", maxUInt);
15     printf("overflow2: %u\n", overflow2);
16
17     return 0;
18 }
```

```
maxInt: 2147483647
overflow1: -2147483648
maxUInt: 4294967295
overflow2: 0
```

2. 자료형

예제 3-4 int 자료형에서 언더플로가 발생하는 경우

```
01 #include <stdio.h>
02
03 int main()
04 {
05     int minInt1 = -2147483648LL;    // int의 최솟값으로 초기화
06     int underflow1 = minInt1 - 1;
07
08     printf("minInt: %d\n", minInt1);
09     printf("underflow1: %d\n", underflow1);
10
11     unsigned int minInt2 = 0;      // 부호 없는 int 0으로 초기화
12     unsigned int underflow2 = minInt2 - 1;
13
14     printf("minInt: %d\n", minInt2);
15     printf("underflow2: %u\n", underflow2);
16
17     return 0;
18 }
```

언더플로가 발생하는 연산

언더플로가 발생하는 연산

```
minInt: -2147483648
underflow1: 2147483647
minInt: 0
underflow2: 4294967295
```

2. 자료형

하나 더 알기 롤오버

C 언어에서 롤오버는 특정한 범위의 한계를 초과하여 다시 시작점으로 돌아가는 현상을 의미한다. 주로 정수형의 오버플로 또는 언더플로 상황에서 발생한다.



그림 3-14 `int` 자료형의 롤오버 작동 방식

2. 자료형

■ 실수형

■ 실수형의 종류와 유효 범위

- **실수형:** 소수점이 있는 수를 저장하기 위한 자료형
- **부동 소수점 방식:** 소수점의 위치를 고정하지 않고, 가수와 실수를 이용하여 실수를 표현하는 방식

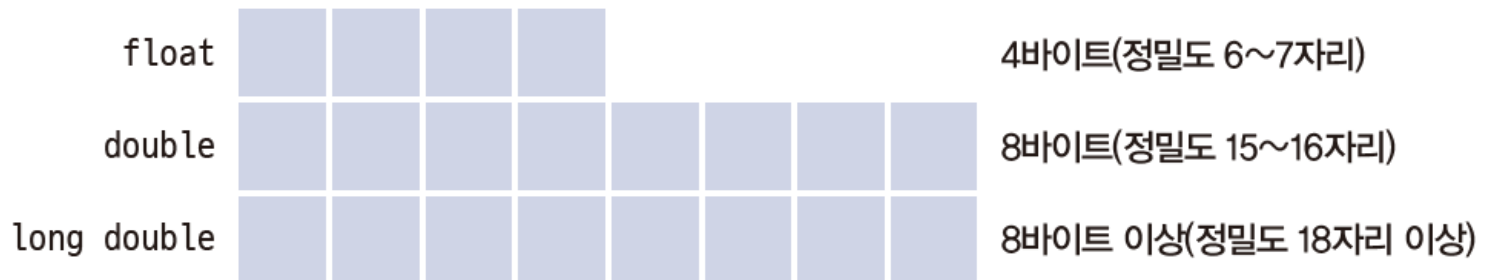


그림 3-15 실수형의 종류와 메모리 저장 공간

2. 자료형

■ 실수형

■ 실수형의 종류와 유효 범위

표 3-3 실수형의 종류와 정밀도, 유효 범위

실수형	크기	정밀도(유효 숫자)	유효 범위
float	4바이트	6~7자리	최솟값: $1.175494351E-38$
			최댓값: $3.402823466E+38$
double	8바이트	15~16자리	최솟값: $\pm 2.2250738585072014E-308$
			최댓값: $\pm 1.7976931348623157E+308$
long double	8바이트 이상	18자리 이상	최솟값: $\pm 3.36210314311209350626E-4932$
			최댓값: $\pm 1.18973149535723176502E+4932$

2. 자료형

예제 3-5 실수형 변수를 사용하여 사칙 연산하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     float floatValue1 = 3.14f;    // float형 변수 선언과 값 할당
06     float floatValue2 = 2.718f;    // float형 변수 선언과 값 할당
07     float sumFloat = floatValue1 + floatValue2;
08     float productFloat = floatValue1 * floatValue2;
09
10     printf("Sum (Float): %f\n", sumFloat);
11     printf("Product (Float): %f\n", productFloat);
12
13     double doubleValue1 = 123.456;    // double형 변수 선언과 값 할당
14     double doubleValue2 = 789.012;    // double형 변수 선언과 값 할당
15     double sumDouble = doubleValue1 + doubleValue2;
16     double productDouble = doubleValue1 * doubleValue2;
17
18     printf("Sum (Double): %lf\n", sumDouble);
19     printf("Product (Double): %lf\n", productDouble);
20
21     return 0;
22 }
```

변수 sumFloat에 두 변수의 덧셈 결과 저장

변수 productFloat에 두 변수의 곱셈 결과 저장

변수 sumDouble에 두 변수의 덧셈 결과 저장

변수 productDouble에 두 변수의 곱셈 결과 저장

```
Sum (Float): 5.858000
Product (Float): 8.534520
Sum (Double): 912.468000
Product (Double): 97408.265472
```

2. 자료형

예제 3-6 실수형 변수를 사용하여 삼각형과 사각형의 넓이 계산하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     float base = 5.0;
06     float height = 3.0;
07     float triangleArea = (base * height) / 2.0;
08
09     printf("삼각형 넓이: %.2f\n", triangleArea);
10
11     double width = 4.0;
12     double rheight = 6.0;
13     double rectangleArea = width * rheight;
14
15     printf("사각형 넓이: %.2lf\n", rectangleArea);
16
17     return 0;
18 }
```

삼각형의 넓이를 계산하는 함수

삼각형의 넓이 출력

사각형의 넓이를 계산하는 함수

사각형의 넓이 출력

```
삼각형 넓이: 7.50
사각형 넓이: 24.00
```

2. 자료형

■ 문자형

- 단일 문자를 저장하기 위한 자료형



그림 3-16 문자형의 종류

표 3-4 문자형의 종류와 유효 범위

문자형		크기	유효 범위
char	signed char	1바이트	-128~127
	unsigned char		0~255

```
char myChar;    // 문자형 변수 선언
myChar = 'A';   // 문자 'A' 저장
```

```
char myChar;    // 문자형 변수 선언
myChar = 65;    // 아스키코드에서 'A'의 값
```

2. 자료형

■ 문자형

■ 아스키코드

- C 언어가 처음 개발되었을 당시에 주로 사용되던 문자 인코딩 체계 중 하나
- C 언어와 아스키코드가 공존하면서 발전했기에 C 언어의 문자와 문자열은 아스키코드로 변환되며, 현재 다른 언어들은 유니코드를 지원

제어 문자			출력 가능 문자									제어 문자			출력 가능 문자											
10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자			
000	00	NULL	032	20	SP	064	40	@	096	60	`	016	10	DLE	048	30	0	080	50	P	112	70	p			
001	01	SOH	033	21	!	065	41	A	097	61	a	017	11	DC1	049	31	1	081	51	Q	113	71	q			
002	02	STX	034	22	*	066	42	B	098	62	b	018	12	DC2	050	32	2	082	52	R	114	72	r			
003	03	ETX	035	23	#	067	43	C	099	63	c	019	13	DC3	051	33	3	083	53	S	115	73	s			
004	04	EOT	036	24	\$	068	44	D	100	64	d	020	14	DC4	052	34	4	084	54	T	116	74	t			
005	05	ENQ	037	25	%	069	45	E	101	65	e	021	15	NAK	053	35	5	085	55	U	117	75	u			
006	06	ACK	038	26	&	070	46	F	102	66	f	022	16	SYN	054	36	6	086	56	V	118	76	v			
007	07	BEL	039	27	'	071	47	G	103	67	g	023	17	ETB	055	37	7	087	57	W	119	77	w			
008	08	BS	040	28	(072	48	H	104	68	h	024	18	CAN	056	38	8	088	58	X	120	78	x			
009	09	HT	041	29)	073	49	I	105	69	i	025	19	EM	057	39	9	089	59	Y	121	79	y			
010	0A	LF	042	2A	*	074	4A	J	106	6A	j	026	1A	SUB	058	3A	:	090	5A	Z	122	7A	z			
011	0B	VT	043	2B	+	075	4B	K	107	6B	k	027	1B	ESC	059	3B	;	091	5B	[123	7B	{			
012	0C	FF	044	2C	.	076	4C	L	108	6C	l	028	1C	FS	060	3C	<	092	5C	\	124	7C				
013	0D	CR	045	2D	-	077	4D	M	109	6D	m	029	1D	GS	061	3D	=	093	5D]	125	7D	}			
014	0E	SOH	046	2E	.	078	4E	N	110	6E	n	030	1E	RS	062	3E	>	094	5E	^	126	7E	~			
015	0F	SI	047	2F	/	079	4F	O	111	6F	o	031	1F	US	063	3F	?	095	5F	_	127	7F	DEL			

2. 자료형

예제 3-7 알파벳의 아스키코드 값 출력하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     char alphabet = 'T';
06
07     printf("alphabet 변수의 값: %c\n", alphabet);
08     printf("알파벳 'T'에 해당하는 ASCII 코드 값: %d\n", alphabet);
09
10     return 0;
11 }
```

문자형 변수 alphabet에 'T' 저장

변수 alphabet의 값 출력

알파벳 'T'에 해당하는 아스키코드 값 출력

```
alphabet 변수의 값: T
알파벳 'T'에 해당하는 ASCII 코드 값: 84
```

Section 3.3

상수

3. 상수

■ 상수의 개념

- 프로그램에서 고정된 값을 나타내는 식별자
- 한 번 정의하면 그 값이 유지
- 정수형 상수
- 실수형 상수
- 문자형 상수
 - 단일 문자를 나타내는 상수로, 작은따옴표(' ')로 표기
- 문자열 상수
 - 여러 개의 문자로 이루어진 문자열을 나타내는 상수로, 큰따옴표("")로 표기

변수 1 상수 변수 2

원의 넓이 = 3.14 × 반지름²

그림 3-17 원의 넓이를 구하는 공식의 상수와 변수

3. 상수

■ 정수형 상수

■ 10진수

```
printf("Decimal: %d\n", 42);
```

10진수 정수를 출력하는 형식 지정자

■ 16진수

```
printf("Hexadecimal: %x\n", 0x2A);
```

16진수 정수를 출력하는 형식 지정자

■ 8진수

```
printf("Octal: %o\n", 052);
```

8진수 정수를 출력하는 형식 지정자

■ 2진수

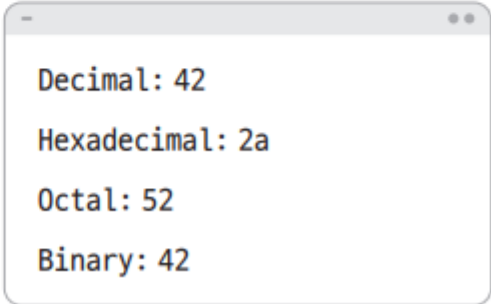
```
printf("Binary: %d\n", 0b101010);
```

10진수 정수를 출력하는 형식 지정자

3. 상수

예제 3-8 10진수, 16진수, 8진수, 2진수 상수 출력하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     int decimal = 42;           // 10진수 상수
06     int hexadecimal = 0x2A;     // 16진수 상수
07     int octal = 052;           // 8진수 상수
08     int binary = 0b101010;     // 2진수 상수
09
10     printf("Decimal: %d\n", decimal);
11     printf("Hexadecimal: %x\n", hexadecimal);
12     printf("Octal: %o\n", octal);
13     printf("Binary: %d\n", binary);
14
15     return 0;
16 }
```



```
Decimal: 42
Hexadecimal: 2a
Octal: 52
Binary: 42
```

3. 상수

■ 실수형 상수

- 부동 소수점 형식으로 표현
- **10진수 상수:** 10진수 형식으로 나타낸 실수
 - [예] 3.14, - 2.5, 0.0
- **16진수 상수:** 16진수 형식으로 나타낸 실수
 - [예] 0x1.8p2는 1.8×2^2 을 나타냄
- **지수 표기법:** e 또는 E를 사용하여 지수를 표기
 - [예] $1.23e - 5$ 는 1.23×10^{-5} 을 나타냄

3. 상수

■ 실수형 상수

■ 지수 형식

- 가수

- 실수의 유효 숫자 부분

- 지수

- 실수를 곱하는 지수를 의미

$$589.734 \rightarrow \boxed{5.89734} \times \boxed{10^2}$$

유효 숫자 가수 부분 지수 부분

그림 3-18 지수 형식의 구성 요소



그림 3-19 지수 형식으로 나타낸 실수형 상수

3. 상수

■ 실수형 상수

▪ float형 상수

- 32비트 부동 소수점 형식으로 표현되는 실숫값

```
float fscore;
```

```
fscore = 589.734F;
```

float형 상수 표현 형식으로 f 또는 F를 사용



그림 3-20 float형 상수의 비트 자릿수 구성

3. 상수

예제 3-9 float형 상수 출력하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     float fscore;
06     fscore = 589.73456F;
07
08     printf("fscore: %f\n", fscore);
09     printf("fscore: %.2f\n", fscore);
10     printf("fscore: %e\n", fscore);
11     printf("fscore: %E\n", fscore);
12
13     return 0;
14 }
```

float형 상수 589.73456

소수점 아래 둘째 자리까지 출력

지수 형식 e로 출력

지수 형식 E로 출력

```
fscore: 589.734558
fscore: 589.73
fscore: 5.897346e+02
fscore: 5.897346E+02
```

3. 상수

■ 실수형 상수

▪ double형 상수

- 64비트 부동 소수점 형식으로 표현되는 실숫값

```
double dscore;  
dscore = 589.73456;
```



그림 3-21 double형 상수의 비트 자릿수 구성

3. 상수

예제 3-10 double형 상수 출력하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     double dscore;
06     dscore = 589.73456;
07
08     printf("dscore: %lf\n", dscore);
09     printf("dscore: %.2lf\n", dscore);
10     printf("dscore: %e\n", dscore);
11     printf("dscore: %E\n", dscore);
12
13     return 0;
14 }
```

double형 상수 589.73456

소수점 아래 여섯째 자리까지 출력

소수점 아래 두 자리 정밀도로 double 값 출력

지수 형식 e로 출력

지수 형식 E로 출력

```
dscore: 589.734560
dscore: 589.73
dscore: 5.897346e+02
dscore: 5.897346E+02
```

3. 상수

■ 문자형 상수

- **알파벳 대문자:** 알파벳 대문자를 나타내는 문자형 상수
 - [예] 'A', 'B', 'C'
- **알파벳 소문자:** 알파벳 소문자를 나타내는 문자형 상수
 - [예] 'a', 'b', 'c'
- **숫자:** 숫자를 나타내는 문자형 상수
 - [예] '0', '1', '2'
- **특수 문자:** 특수 문자를 나타내는 문자형 상수
 - [예] '\$', '%', '@'

3. 상수

예제 3-11 문자형 상수를 아스키코드로 출력하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     char grade = 'A';
06     char dollar = '$';
07     char digit = '7';
08
09     printf("Grade 값: %c\n", grade);
10     printf("Grade 아스키코드: %d\n", grade);
11
12     printf("Dollar Sign 값: %c\n", dollar);
13     printf("Dollar Sign 아스키코드: %d\n", dollar);
14
15     printf("Digit 값: %c\n", digit);
16     printf("Digit 아스키코드: %d\n", digit);
17
18     return 0;
19 }
```

문자를 출력하는 형식 지정자

문자 '7'에 해당하는 아스키코드 값 출력

```
Grade 값: A
Grade 아스키코드: 65
Dollar Sign 값: $
Dollar Sign 아스키코드: 36
Digit 값: 7
Digit 아스키코드: 55
```

3. 상수

■ 기호 상수

- 반복적으로 사용하는 값이나 매직 넘버를 대체하기 위해 이름으로 정의된 상수
- 전역상수로 선언되며, 일반적으로 대문자와 언더바(_)를 사용하여 이름 작성

■ #define 지시문을 사용하여 정의

| Syntax | #define 지시문

지시문 기호 상수 상숫값

```
#define PI 3.14159
```

■ const 명령어를 사용하여 정의

| Syntax | const 명령어

명령어 자료형 기호 상수 상숫값;

```
const double PI 3.14159;
```

3. 상수

하나 더 알기 매직 넘버

C 언어에서 매직 넘버(magic number)는 소스 코드 내에서 문맥 없이 사용된 특정 숫자를 의미한다. 이러한 숫자는 코드의 명확성을 감소시키며, 다른 사람들이 코드를 읽을 때 그 숫자가 무엇을 의미하는지, 왜 그러한 값이 선택되었는지 알기 어렵다. 다음 코드를 살펴보자.

```
if (salary > 40000)
{
    // do something
}
```

매직 넘버

여기서 숫자 '40000'은 매직 넘버로, 코드만 보아서는 이 숫자의 의미가 명확하지 않다. 이럴 때 다음과 같이 UPPER_LIMIT_SALARY라는 기호 상수를 사용하면 해당 숫자가 최대 급여 한도를 의미한다는 것을 쉽게 알 수 있다.

```
const int UPPER_LIMIT_SALARY = 40000;
```

기호 상수를 정의하는 구문

```
if (salary > UPPER_LIMIT_SALARY)
{
    // do something
}
```

숫자 대신 기호 상수 입력

이처럼 매직 넘버 대신 의미 있는 이름의 상수나 변수를 사용하면 코드를 직관적으로 이해할 수 있어 중요한 값이 변경되는 것을 방지함으로써 오류가 발생할 가능성도 줄어든다.

3. 상수

예제 3-12 #define 지시문으로 기호 상수 정의하고 활용하기

```
01 #include <stdio.h>
```

```
02
```

```
03 #define PI 3.14159265
```

#define 지시문으로 기호 상수 정의

```
04
```

```
05 int main()
```

```
06 {
```

```
07     float radius = 5.0f;
```

```
08     float area = 0;
```

```
09
```

```
10     area = PI * radius * radius;
```

기호 상수 PI를 이용한 연산 코드

```
11
```

```
12     printf("반지름: %.2f\n", radius);
```

소수점 아래 여덟째 자리까지 출력

```
13     printf("기호 상수 PI: %.8f\n", PI);
```

```
14     printf("원의 넓이: %.8f\n", area);
```

```
15
```

```
16     return 0;
```

```
17 }
```

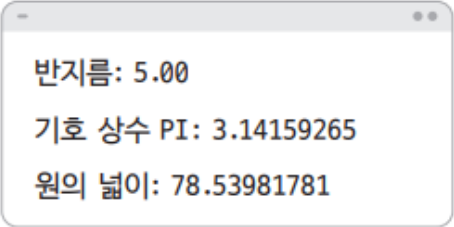
```
반지름: 5.00
기호 상수 PI: 3.14159265
원의 넓이: 78.53981781
```

3. 상수

예제 3-13 `const` 명령어로 기호 상수 정의하고 활용하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     const double PI = 3.14159265;
06
07     double radius = 5.0;
08     double area = 0;
09
10     area = PI * radius * radius;
11
12     printf("반지름: %.2lf\n", radius);
13     printf("기호 상수 PI: %.8lf\n", PI);
14     printf("원의 넓이: %.8lf\n", area);
15
16     return 0;
17 }
```

`const` 명령어로 기호 상수 선언



```
반지름: 5.00
기호 상수 PI: 3.14159265
원의 넓이: 78.53981781
```

Section 3.4

자료형 변환

4. 자료형 변환

■ 자료형 변환의 정의

- 어떤 자료형의 데이터를 다른 자료형으로 변환하는 과정
- C 언어에서 변수를 선언하고 프로그램을 실행하다 보면 연산 처리 과정에서 원치 않는 결괏값이 산출되는 경우
 - 예를 들어 정수와 정수의 나눗셈 연산을 수행하여 실수형 변수에 결괏값을 대입했을 때 실수가 아닌 정수가 실행 결과로 출력 가능

4. 자료형 변환

■ 자료형 변환의 정의

정수와 정수의 나눗셈 연산

Division.c

```
01 #include <stdio.h>
02
03 int main()
04 {
05
06     int a = 5;    // int형 변수 a를 선언하고 5로 초기화
07     int b = 3;    // int형 변수 b를 선언하고 3으로 초기화
08     float c = a / b;
09
10     printf("%d / %d = %f \n", a, b, c);
11
12     return 0;
13 }
```

5 / 3 = 1.000000

정수 부분만을 반환(소수점 공간이 없음)

4. 자료형 변환

■ 자동 형 변환

- 특정 연산에서 다른 자료형이 함께 사용될 때 컴파일러가 크기가 작은 자료형에서 크기가 큰 자료형으로 자동으로 변환



그림 3-22 자동 형 변환의 순서

```
int i = 10;           // int형 변수 i를 선언하고 10으로 초기화
float f = 2.5;        // float형 변수 f를 선언하고 2.5로 초기화
float result = i * f;  // i가 float형으로 자동으로 변환됨
```

- ① 먼저 int형의 값이 데이터 크기가 더 큰 float형으로 자동으로 변환된다.
- ② 따라서 두 float 값의 곱셈 연산이 수행되어 결과값이 float형으로 반환된다.

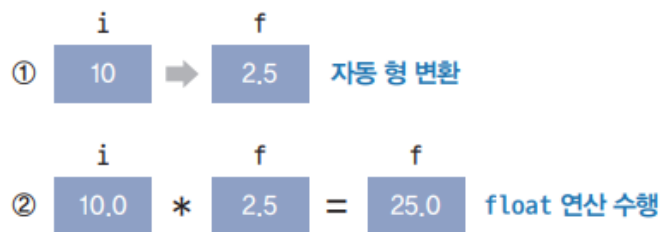


그림 3-23 자동 형 변환의 연산 과정

4. 자료형 변환

■ 명시적 형 변환

- 프로그래머가 직접 자료형 변환을 지시하는 방식

| Syntax | 형 변환 연산자

(자료형)피연산자

(int)2.78

```
int i = 10;           // int형 변수 i를 선언하고 10으로 초기화
float f = 2.5;        // float형 변수 f를 선언하고 2.5로 초기화
int result = (int)f * i; // 형 변환 연산자를 사용하여 변수 f를 int형으로 변환
```

4. 자료형 변환

예제 3-14 자동 형 변환과 명시적 형 변환으로 연산 수행하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     int i = 5;
06     double d = 2.2;
07     double result1 = i + d;
08     double result2 = i * d;
09     double result3 = (double)i - d;
10     double result4 = (double)i / d;
11
12     printf("덧셈 결과: %lf\n", result1);
13     printf("곱셈 결과: %lf\n", result2);
14     printf("뺄셈 결과: %lf\n", result3);
15     printf("나눗셈 결과: %lf\n", result4);
16     return 0;
17 }
```

연산 과정에서 int형 변수 i가
double형으로 자동 형 변환

형 변환 연산자를 이용하여 int형 변수
i를 double형으로 명시적 형 변환

```
덧셈 결과: 7.200000
곱셈 결과: 11.000000
뺄셈 결과: 2.800000
나눗셈 결과: 2.272727
```

4. 자료형 변환

예제 3-15 double형에서 int형으로 자동 형 변환 수행하기

```
01 #include <stdio.h>
```


```
02
```


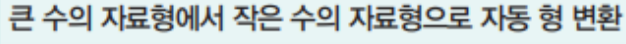
```
03 int main()
```

```
04 {
```

```
05     double d = 5.0;
```

```
06     int i;
```

```
07      double형 변수 d의 값을 int형 변수 i에 대입
```

```
08      i = d;  큰 수의 자료형에서 작은 수의 자료형으로 자동 형 변환
```


```
09
```

```
10     printf("i의 값: %d\n", i);
```

```
11
```

```
12     return 0;
```

```
13 }
```

i의 값: 5  소수 부분이 제거되고 정수 부분만 반환

4. 자료형 변환

예제 3-16 double형에서 int형으로 명시적 형 변환 수행하기

```
01 #include <stdio.h>
02
03 int main()
04 {
05     double d = 5.0;
06     int i;
07
08     i = (int)d;
09
10     printf("i의 값: %d\n", i);
11
12     return 0;
13 }
```

double형 변수 d의 값을 int형으로 명시적
형 변환을 한 후 그 결과를 변수 i에 할당

큰 수의 자료형에서 작은 수의
자료형으로 명시적 형 변환

i의 값: 5 소수 부분이 제거되고 정수 부분만 반환

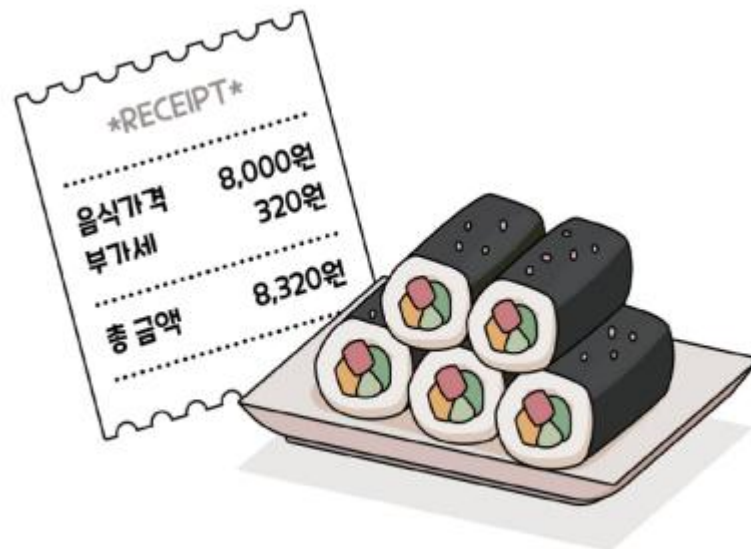
4. 자료형 변환

LAB 음식 가격에 부가세 적용하기

■ 실행 결과

음식의 가격을 입력하세요(원): 8000

부가세 포함 총가격: 8320.00원



4. 자료형 변환

LAB 음식 가격에 부가세 적용하기

■ 조건

- 음식 가격을 입력받는다.
- 입력받은 음식 가격에 4%의 부가세를 덧붙여 총가격을 계산한다.
- 부가세가 포함된 총가격을 소수점 아래 둘째 자리까지 화면에 출력한다.

■ 힌트

- 상수를 사용하여 부가세율을 0.04로 정의한다.
- 부가세가 포함된 총가격은 입력받은 음식 가격에 부가세율을 곱한 값을 더하여 계산한다.
- printf() 함수의 형식 지정자를 사용하여 소수점 아래 둘째 자리까지 출력한다.
- 음식 가격을 입력받아 변수를 선언하고, 이 변수에 부가세율을 적용하여 총가격을 계산한다. 단, 음식 가격을 입력받기 전 변수의 초기값을 확인해야 한다.

4. 자료형 변환

LAB 음식 가격에 부가세 적용하기

문제 해결

LAB_3-1.c

```
01 #pragma warning(disable: 4996)
02 #include <stdio.h>
03 #define VAT_RATE 0.04 // 상수 선언, 부가세율 0.04
04                        // 부가세율을 나타내는 상수 정의
05 int main()
06 {
07     double foodPrice; // 음식 가격 초기화
08     double totalPrice = 0.0; // 총가격 초기화
09                        // 부가세가 포함된 가격을 나타내는 변수 totalPrice 선언 및 초기화
10     printf("음식의 가격을 입력하세요(원): ");
11     scanf("%lf", &foodPrice);
12                        // 부동 소수점 수를 입력받기 위한 형식 지정자
13     totalPrice = foodPrice + (foodPrice * VAT_RATE); // 총가격 계산
14
15     printf("부가세 포함 총가격: %.2lf원\n", totalPrice);
16                        // 소수점 아래 둘째 자리까지 출력하는 형식 지정자
17     return 0;
18 }
```

4. 자료형 변환

LAB 전기 요금 계산하기

■ 실행 결과

```
사용한 전력량(kW)을 입력하세요: 150
전력 요금(1kW당 비용)을 입력하세요: 20
전기 요금: 3000
```



■ 조건

- 전력 사용량(kW)과 전력 요금(1kW당 비용)을 정수형으로 입력받는다.
- 전력 요금을 계산할 때 오버플로를 방지하기 위해 int형보다 큰 범위의 정수를 저장할 수 있도록 형 변환을 수행한다.

■ 힌트

- 전기 요금은 전력 사용량과 전력 요금을 곱하여 계산한다.
`long long 전기 요금 = (long long)전력 사용량 * 전력 요금;`
- `printf()` 함수의 형식 지정자 `%lld`를 사용하여 전기 요금을 출력한다.

4. 자료형 변환

LAB 전기 요금 계산하기

문제 해결

LAB_3-2.c

```
01 #pragma warning(disable: 4996)
02 #include <stdio.h>
03
04 int main()
05 {
06     int powerConsumed, costPerkW;    // 정수형 변수 선언
07
08     printf("사용한 전력량(kW)을 입력하세요: ");
09     scanf("%d", &powerConsumed);
10     printf("전력 요금(1kW당 비용)을 입력하세요: ");
11     scanf("%d", &costPerkW);
12
13     long long totalCost = (long long)powerConsumed * costPerkW;
14
15     printf("전기 요금: %lld\n", totalCost);
16
17     return 0;
18 }
```

전력 사용량과 전력 요금을 입력받아
저장하기 위한 형식 지정자 %d

오버플로를 방지하기 위한 명시적 형 변환

long long 자료형에 대한 형식 지정자 %lld