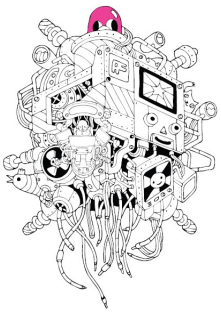


Chapter 05. 상수와 기본 자료형

C언어가 제공하는 기본 자료형의 이해

자료형은 데이터를 표현하는 방법

- 실수를 저장할 것이냐? 정수를 저장할 것이냐!
  - 값을 저장하는 방식이 실수냐 정수냐에 따라서 달라지기 때문에 용도를 결정해야 한다.
- 얼마나 큰 수를 저장할 것이냐!
  - 큰 수를 표현하기 위해서는 많은 수의 바이트가 필요하다.

이름 이외에 메모리 공간의 할당에 있어서 필요한 두 가지 정보

요청의 예

"아! 제가 정수를 저장할건데요. 크기는 4바이트로 하려고 합니다. 그 정도면 충분할거에요. 그리고 변수의 이름은 num으로 할게요.."

제대로 된 요청

C언어에서의 예

```
int num;
```

동일한 요청

- 자료형의 수는 데이터 표현방법의 수를 뜻한다.
- C 언어가 제공하는 기본 자료형의 수가 10개라면, C 언어가 제공하는 기본적인 데이터 표현방식의 수는 10개라는 뜻이 된다.

3

변수의 자료형(Data Type)

- 숫자를 저장하기 위한 변수
  - 정수형 변수와 실수형 변수
- 정수형 변수
  - 정수 값의 저장을 목적으로 선언된 변수
  - 정수형 변수는 크기에 따라 char형, short형, int형, long형 변수로 나뉜다.
- 실수형 변수
  - 실수 값의 저장을 목적으로 선언된 변수
  - 실수형 변수는 float형 변수와 double형 변수로 나뉜다.
- 정수형 변수와 실수형 변수가 나뉘는 이유는?
  - 정수를 저장하는 방식과 실수를 저장하는 방식이 다르기 때문 (Chapter 4)

```
int num1=24;
   · num1은 정수형 변수 중 int형 변수
double num2=3.14;
   · num2는 실수형 변수 중 double형 변수
```

기본 자료형의 종류와 데이터의 표현범위

자료형		크기	값의 표현범위
정수형	char	1바이트	−128이상 +127이하
	short	2바이트	−32,768이상 +32,767이하
	int	4바이트	−2,147,483,648이상 +2,147,483,647이하
	long	4바이트	−2,147,483,648이상 +2,147,483,647이하
	long long	8바이트	−9,223,372,036,854,775,808이상
			+9,223,372,036,854,775,807이하
실수형	float	4바이트	$\pm 3.4\times10^{-37}$이상 $\pm 3.4\times10^{+38}$이하
	double	8바이트	$\pm 1.7\times10^{-307}$이상 $\pm 1.7\times10^{+308}$이하
	long double	8바이트 이상	double 이상의 표현범위

- 컴파일러에 따라서 약간의 차이를 보인다.
  - C의 표준에서는 자료형별 상대적 크기를 표준화 할 뿐 구체적인 크기까지 언급하지는 않는다.
- 크게 정수형과 실수형으로 나뉘다.
  - 데이터를 표현하는 방식이 정수형과 실수형 두 가지로 나뉘므로!
- 정수형에도 실수형에도 둘 이상의 기본 자료형이 존재한다.
  - 표현하고자 하는 값의 크기에 따라서 적절히 선택할 수 있도록 다수의 자료형이 제공!

5

정수형 자료형

자료형(data type)		할당되는 메모리 크기	표현 가능한 데이터의 범위
정수형	char	1 바이트	-128 ~ +127
	short	2 바이트	-32,768 ~ +32,767
	int	4 바이트	-2,147,483,648 ~ +2,147,483,647
	long	4 바이트	-2,147,483,648 ~ +2,147,483,647

- 자료형의 크기를 확인하는 방법
  - sizeof() : 변수 또는 자료형의 메모리 크기 값을 알려줌
  - Ex. sizeof (int), sizeof (double),
  - int num = 30; printf("size of num = %d ", sizeof num);
  - sizeof num or sizeof (num) : OK
  - However, sizeof double or sizeof int : X

연산자 sizeof를 이용한 바이트 크기의 확인

```
int main(void)
{
    int num = 10;
    int sz1 = sizeof(num);
    int sz2 = sizeof(int);
    . . . .
}
```

변수 num과 int의 크기를 계산, 그 결과로 sz1과 sz2 초기화

- **sizeof 연산자의 피연산자로는 변수, 상수 및 자료형의 이름 등이 올 수 있다.**
- **소괄호는 int와 같은 자료형의 이름에만 필수!**
- **하지만 모든 피연산자를 대상으로 소괄호를 감싸주는 것이 일반적!**

```
int main(void)
{
    char ch=9;
    int inum=1052;
    double dnum=3.1415;
    printf("변수 ch의 크기: %d \n", sizeof(ch));
    printf("변수 inum의 크기: %d \n", sizeof(inum));
    printf("변수 dnum의 크기: %d \n", sizeof(dnum));

    printf("char의 크기: %d \n", sizeof(char));
    printf("int의 크기: %d \n", sizeof(int));
    printf("long의 크기: %d \n", sizeof(long));
    printf("long long의 크기: %d \n", sizeof(long long));
    printf("float의 크기: %d \n", sizeof(float));
    printf("double의 크기: %d \n", sizeof(double));
    return 0;
}
```

SizeOfOperator.c

실행결과

```
변수 ch의 크기: 1
변수 inum의 크기: 4
변수 dnum의 크기: 8
char의 크기: 1
int의 크기: 4
long의 크기: 4
long long의 크기: 8
float의 크기: 4
double의 크기: 8
```

7

정수의 표현 및 처리를 위한 일반적 자료형 선택

- **정수형 자료형 데이터 연산 과정**

```
short num3 = 100, num4 = 200;
short result = 0;

result = num3 + num4
printf("Size of short addition : %d ₩n", sizeof(num3 + num4));
printf("Size of result : %d ₩n", sizeof(result));
```

CharShortBaseAdd.c

- **일반적으로 가장 많이 사용하는 자료형은 int이다.**
  - CPU가 연산하기에 가장 적합한 데이터의 크기가 int형의 크기로 결정된다.
  - 따라서 int형 연산속도가 다른 자료형 연산속도와 동일하거나 빠름!!
- **char형 short형 변수는 불필요한가?**
  - 데이터의 크기가 char 또는 short로 충분히 표현 가능하다면, char 또는 short로 데이터를 표현 및 저장하는 것이 적절

프로그램 연습

- 정수를 저장하는 방을 두 개 (room1, room2) 만들어 각각 100과 200을 저장하고 그 값들의 합, 차, 곱을 출력하는 프로그램을 작성하시오.
  - 단, 위의 요구만을 만족하기 위해 필요한 메모리를 가장 적게 차지하도록 방을 만들도록 하시오.

```
room1과 room2의 합은 300입니다.
room1과 room2의 차는 -100입니다.
room1과 room2의 곱은 20000입니다.
```

부동 소수형 (실수형)

- float, double, long double
  - float, double, long double의 순으로 표현 범위가 크고, 세밀함

```
float  <  double  <=  long double
```

  - float의 저장공간 크기 : 32비트(4바이트) ➔ 대략 10^{-38}에서 10^{38}
  - double의 저장공간 크기 : 64비트(8바이트) ➔ 대략 10^{-308} 에서 10^{308}
  - long double : 컴파일러마다 다름
    - Visual C++에서는 long double도 double과 마찬가지로 8바이트

실수의 표현 및 처리를 위한 일반적 자료형 선택

- 실수 자료형의 선택기준은 정밀도
  - 실수의 표현범위는 float, double 둘 다 충분히 넓다.
  - 그러나 8바이트 크기의 double이 float보다 더 정밀하게 실수를 표현
- 일반적인 선택은 double이다.
  - 컴퓨팅 환경의 발전으로 double형 데이터의 표현 및 연산이 덜 부담스럽다.
  - float형 데이터의 정밀도는 부족한 경우가 많다.

실수 자료형	소수점 이하 정밀도	바이트 수
float	6자리	4
double	15자리	8
long double	18자리	12

```
int main(void)
{
    double rad;
    double area;
    printf("원의 반지름 입력: ");
    scanf("%lf", &rad);
    area = rad*rad*3.1415;
    printf("원의 넓이: %f \n", area);
    return 0;
}
```

double형 변수의 출력 서식문자 %f, %lf - printf

double형 변수의 입력 서식문자 %lf - scanf

실행결과

```
원의 반지름 입력: 2.4
원의 넓이: 18.095040
```

CircleArea.c

부동 소수형 (실수형)

- float형, double형
  - float형 변수의 선언, 값의 저장 및 출력

```
#include <stdio.h>
int main(void)
{
        float a = 3.5, b = 2.2, c;
        double d;
        d = 100.94;
        c = a + b;
        d = d * 2;
        printf("a+b 결과는 %f입니다.₩n", c);
        printf("d의 값은 %lf입니다.₩n", d);
        return 0;
}
```

```
float a = 3.5;
float b = 2.2;
float c;
```

Unsigned를 붙여서 0과 양의 정수만 표현

- 정수 자료형의 이름 앞에 **unsigned** 선언을 붙일 수 있음
- **unsigned** 선언이 붙으면 MSB도 데이터의 크기를 표현하는데 사용됨
- 따라서 표현하는 값의 범위가 **양의 정수**로 제한되어 양의 정수 값이 2배로 늘어남

정수 자료형	크기	값의 표현범위
char	1바이트	−128이상 +127이하
unsigned char		0이상 (128 + 127)이하
short	2바이트	−32,768이상 +32,767이하
unsigned short		0이상 (32,768 + 32,767)이하
int	4바이트	−2,147,483,648이상 +2,147,483,647이하
unsigned int		0이상 (2,147,483,648 + 2,147,483,647)이하
long	4바이트	−2,147,483,648이상 +2,147,483,647이하
unsigned long		0이상 (2,147,483,648 + 2,147,483,647)이하
long long	8바이트	−9,223,372,036,854,775,808이상 +9,223,372,036,854,775,807이하
unsigned long long		0이상 (9,223,372,036,854,775,808 + 9,223,372,036,854,775,807)이하

프로그램 연습

- 아래의 지시에 따라 순서대로 프로그램 하시오.
  1) 정수형 변수 a와 실수형 변수 b, double형 (큰 실수형) 변수 c를 선언
  2) 실수형 변수 d와 f를 선언하면서 각각 3.5와 2.2를 저장
  3) d와 f의 곱을 a에 저장하고 출력
  4) d와 f의 곱을 b에 저장하고 출력
  5) d와 f의 곱을 c에 저장하고 출력

  - 원하는 결과가 나오는지 확인

```
d*f의 결과는 a에 저장한 결과는 7입니다.
d*f의 결과를 b에 저장한 결과는 7.700000입니다.
d*f의 결과를 c에 저장한 결과는 7.700000입니다.
```

문자의 표현방식과 문자를 위한 자료형

문자 변수 선언

- 하나의 영문자 저장을 위한 변수 선언
  - char c;
  - 영문자 하나 (1 바이트)를 저장하기 위해 char형 변수 c를 선언
  - 1 바이트 (8비트) 정수를 저장하기 위해서도 선언 가능

```
#include <stdio.h>

int main(void)
{
        char c;
        c = 'A';
        c = 'a';

        printf("변수 c에 들어있는 문자는 %c입니다.₩n", c);

        return 0;
}
```

문자의 표현을 위한 약속! 아스키(ASCII) 코드!

아스키 코드	아스키 코드 값
A	65
B	66
C	67
`	96
~	126

미국 표준 협회(ANSI: American National Standards Institute)에 의해서 제정된 아스키(ASCII: Amerian Standard Code for Information Interchange) 코드

- 컴퓨터는 문자를 표현 및 저장하지 못한다.
- 따라서 문자를 표현을 목적으로 각 문자에 고유한 숫자를 지정한다.
- 인간이 입력하는 문자는 해당 문자의 숫자로 변환이 되어 컴퓨터에 저장 및 인식
- 컴퓨터에 저장된 숫자는 문자로 변환이 되어 인간의 눈에 보여지게 됨

```
int main(void)
{
    char ch1 = 'A';
    char ch2 = 'C';
    . . . .
}
```

```
int main(void)
{
    char ch1 = 65;
    char ch2 = 67;
    . . . .
}
```

- 컴파일 시 각 문자는 해당 아스키 코드 값으로 변환
- 따라서 실제로 컴퓨터에게 전달되는 데이터는 문자가 아닌 숫자

C 프로그램상에서 문자는 작은 따옴표로 묶어서 표현

문자는 이렇게 표현되는 거구나!

```
int main(void)
{
    char ch1='A', ch2=65;
    int ch3='Z', ch4=90;

    printf("%c %d \n", ch1, ch1);
    printf("%c %d \n", ch2, ch2);
    printf("%c %d \n", ch3, ch3);
    printf("%c %d \n", ch4, ch4);
    return 0;
}
```

HowChar.c

서식문자 %c 해당 숫자의 아스키 코드 문자를 출력해라!

```
A 65
A 65
Z 90
Z 90
```

실행결과

- **문자를 char형 변수에 저장하는 이유**
  - 모든 아스키 코드 문자는 1바이트로도 충분히 표현가능
  - 문자는 덧셈, 뺄셈과 같은 연산을 동반하지 않는다. 단지 표현에 사용될 뿐
  - 따라서 1바이트 크기인 char형 변수가 문자를 저장하기 최적의 장소
  - 문자는 int형 변수에도 저장이 가능

아스키 code table

- **ASCII code**

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

  - 'A'는 실제로는 01000001 (= 십진수로 65)로 저장됨
  - 'B'는 실제로는 01000010 (= 십진수로 66)로 저장됨
  - 가장 큰 수는 01111111 (DEL)로 십진수로는 127임 (첫 bit는 항상 0)

문자 코드

- **아스키코드 이용**

문자이름	영문 표현	정수값	\nnn	표현	의미
경고	BEL (Bell)	7	\007	\a	경고음이 울림
수평탭	HT (Horizontal tab)	9	\011	\t	수평으로 다음 탭만큼 이동
개행문자	LF (Line feed)	10	\012	\n	다음 줄로 이동
폼피드	FF (Form feed)	12	\014	\f	새 페이지의 처음으로 이동
캐리지 리턴	CR (Carriage return)	13	\015	\r	현재 줄의 처음으로 이동
큰따옴표	Double quote	34	\042	\"	" 문자
작은따옴표	Single quote	39	\047	\'	' 문자
역슬래쉬	Backslash	92	\134	\\	\ 문자

- **출력제어문자**
  - char alarm = '\a'; 또는 char alarm = 7;
  - char backslash = 92; 또는 char backslash = '\\'; // 92는 1011100
  - char backslash1 = '\\';

확장 아스키 code table

- **unsigned** char c; 이용해야 함

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	⌂

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	|	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	ø	187	BB	╗	219	DB	▉	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	⌂

문자형의 표현

- 문자의 내부 값
  - 메모리에는 문자가 저장되는 것이 아니라 문자에 대응되는 정수 (ASCII 값) 저장

```
char  c = 'a';
```

  - 따라서 문자형 자료형은 넓은 의미로 정수형 자료형
  - 문자형 변수는 내부적으로는 정수형이므로 산술연산이 가능
    - 문자 'A'의 다음 두 번째 문자인 'B'가 출력

```
char upper = 'A';
printf("대문자 A 출력 > %c₩n", upper);
printf("대문자 A 출력 > %d₩n", upper);
printf("대문자 B 출력 > %c₩n", upper+1);
printf("대문자 B 출력 > %d₩n", upper+1);
```

13가지 자료형 (1/2)

- 기본 자료형

자료형(data type)		할당되는 메모리 크기	표현 가능한 데이터의 범위
정수형	char	1 바이트	-128 ~ +127
	short	2 바이트	-32768 ~ +32767
	int	4 바이트	-2147483648 ~ +2147483647
	long	4 바이트	-2147483648 ~ +2147483647
실수형	float	4 바이트	$3.4*10^{-37}$ ~ $3.4*10^{+38}$
	double	8 바이트	$1.7*10^{-307}$ ~ $1.7*10^{+308}$
	long double	8 바이트 혹은 그 이상	차이를 많이 보임

13가지 자료형 (2/2)

- unsigned가 붙어서 달라지는 표현의 범위
  - MSB까지도 데이터의 크기를 표현하는데 사용
  - 양의 정수로 인식
  - 정수형, 문자형에만 unsigned 가능 (실수형 자료형에는 사용 불가)

정수 자료형	크기	값의 표현범위
char	1바이트	−128이상 +127이하
unsigned char		0이상 (128 + 127)이하
short	2바이트	−32,768이상 +32,767이하
unsigned short		0이상 (32,768 + 32,767)이하
int	4바이트	−2,147,483,648이상 +2,147,483,647이하
unsigned int		0이상 (2,147,483,648 + 2,147,483,647)이하
long	4바이트	−2,147,483,648이상 +2,147,483,647이하
unsigned long		0이상 (2,147,483,648 + 2,147,483,647)이하
long long	8바이트	−9,223,372,036,854,775,808이상 +9,223,372,036,854,775,807이하
unsigned long long		0이상 (9,223,372,036,854,775,808 + 9,223,372,036,854,775,807)이하

프로그램 연습

- **자료형 long과 unsigned 확인**
  1) 자료형 long과 unsigned인 방을 각각 만들고 그 방들에 각각 적당한 크기의 값을 저장하고 각 값을 출력 (출력양식: %u)
  2) 이번에는 두 개의 방에 큰 값을 저장하고 각 값을 출력
  3) 값이 제대로 출력되는 범위를 확인하고 어떤 값부터 어떻게 이상하게 나오는 지 확인 (저장 범위 기반)
- **자료형 char 확인**
  1) 자료형 char에 문자 'Q'를 저장하고 문자와 정수 값을 각각 출력
  2) 출력된 정수 값은 교재의 ASCII 값과 동일한지 확인
- 내용
  - 자료형 long과 unsigned는 그 저장 범위인 21억을 벗어난다면 오버플로우가 발생

프로그램 연습 (Lab 1)

- 아래의 지시에 따라 프로그램 하시오.
  1) 프로그램 사용자로부터 알파벳 문자 하나를 입력 받는다.
  2) 입력 받은 알파벳 문자의 아스키코드 값을 출력한다.
  3) 예를 들면, 아래와 같은 형태로 구현한다.

```
알파벳 문자 하나를 입력:
입력받은 알파벳 문자의 아스키 코드 값은 65입니다.
```

26

프로그램 연습 (Lab 2)

- 아래의 지시에 따라 프로그램 하시오.
  1) 두 개의 실수를 입력 받아서 double 변수에 저장하라.
  2) 그리고 두 수의 사칙연산(덧셈, 뺄셈, 곱셈, 나눗셈) 결과를 출력하라.
  3) 예를 들면 아래와 같은 형태로 출력되어야 함 (printf("..")의 이중 따옴표 안에 숫자가 포함되어 있으면 안됨)

```
두 개의 실수를 입력:
35 + 27.5 =
35 - 27.5 =
35 * 27.5 =
35 / 27.5 =
```

27

실습 시간 (2019년 4월 3일)

- 예제 문제 (4개)
  - SizeOfOperator.c, CharShortBaseAdd.c, CircleArea.c
  - HowChar.c
- 실습 문제
  - Lab 1, Lab 2

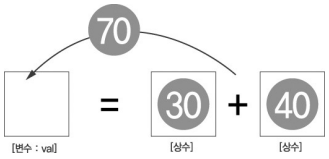
28

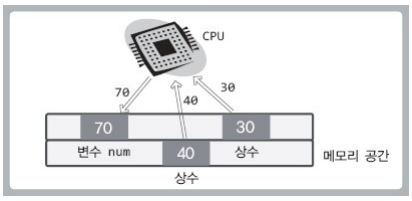
상수에 대한 이해

이름을 지니지 않는 리터럴 (literal) 상수!

- **리터럴 상수: 이름을 지니지 않는 상수**

```
int main(void)
{
    int num = 30 + 40;
    . . . .
}
```





- 단계 1. 정수 30과 40이 메모리 공간에 상수의 형태로 저장된다.
- 단계 2. 두 상수를 기반으로 덧셈이 진행된다.
- 단계 3. 덧셈의 결과로 얻어진 정수 70이 변수 num에 저장된다.

메모리 공간에 저장이 되어야 CPU의 연산대상이 된다.

30

접미사를 이용한 다양한 상수 표현

- 리터럴 상수의 기본 자료형

```
char c = 'A';          // 문자 상수
int I = 5;             // 정수 상수
double d = 3.15;       // 실수 상수

float f = 3.14;        // float f = 3.14f
```

LiteralSize.c

Warning:: 초기화할 때, double형 데이터를 float형 변수에 저장하였으니, 데이터가 잘려나갈 수도 있습니다.

- 접미사를 사용한 다양한 상수 표현

접미사	자료형	사용 예
u or U	unsigned int	unsigned int n = 1025U
l or L	long	long n = 304L
ul or UL	unsigned long	unsigned long n = 304UL
LL	long long	long long n = 5768LL
ULL	unsigned long long	unsigned long long n = 8979ULL
f or F	float	float f = 3.15F
l or L	long double	long double f = 3.15L

이름을 지니는 심볼릭(Symbolic) 상수: const 상수

```
int main(void)
{
    const int MAX=100;    // MAX는 상수! 따라서 값의 변경 불가!
    const double PI=3.1415;    // PI는 상수! 따라서 값의 변경 불가!
    . . . .
}
```

```
int main(void)
{
    const int MAX;    // 쓰레기 값으로 초기화 되어버림
    MAX=100;   // 값의 변경 불가! 따라서 컴파일 에러 발생!
    . . . .
}
```

- **상수의 이름은 모두 대문자로 표시하는 것이 관례!**
- **둘 이상의 단어를 연결할 때에는 MY_AGE와 같이 언더바를 이용해서 두 단어를 구분하는 것이 관례!**

32

실습 문제(Lab3)

- 아래의 지시에 따라 프로그램 하시오.
  1) 정수 (int)형 변수 a를 만든 후 여기에 100을 저장
  2) 작은 정수 (short)형 변수 b를 만든 후 여기에 a의 값을 저장
  3) 실수 (float)형 변수 c를 만든 후 여기에 3.54를 저장
  4) double형 변수 d를 생성과 동시에 456.789를 저장 (const 설정)
  5) sizeof 함수를 이용해서 각 변수 (방)의 크기를 출력
  6) 변수 d의 값을 임의의 다른 값으로 변경해 볼 것
  - 아래와 같은 출력이 되도록 한다. 단, 아래의 숫자들은 printf 함수 호출 시 보이지 않아야 함.

```
a의 값은 100이고, c의 값은 3.540000이고,
d의 값은 456.789000이고, b의 값은 100입니다.
a의 크기는 4, c의 크기는 4, d의 크기는 8,
b의 크기는 2 바이트 입니다.
```

33

자료 형의 변환

자료 형 변환 방식

- 자동 형 변환
  - 대입 과정에서 발생하는 자동 형 변환
  - 연산 수행 시 자동 형 변환
  - 정수의 승격에 의한 자동 형 변환
- 수동 형 변환
  - 강제로 형 변환 수행

35

대입 연산의 과정에서 발생하는 자동 형 변환

```
double num1=245;    // int형 정수 245를 double형으로 자동 형 변환
int num2=3.1415;    // double형 실수 3.1415를 int형으로 자동 형 변환
```

대입 연산자의 왼편을 기준으로 형 변환 발생

정수 245는 245.0의 비트 열로 재구성이 되어 변수 num1에 저장
실수 3.1415는 int형 데이터 3의 비트 열로 재구성이 되어 변수 num2에 저장

```
int num3=129;
char ch=num3;    // int형 변수 num3에 저장된 값이 char형으로 자동 형 변환
```

4바이트 변수 num3에 저장된 4바이트 데이터 중 상위 3바이트가 손실되어 변수 ch에 저장

00000000 00000000 00000000 10000001 ➡ 10000001

36

자동 형 변환의 방식 정리

- 형 변환의 방식에 대한 유형별 정리
  - 정수를 실수로 형 변환 : 3은 3.0으로 5는 5.0으로 변경 (오차가 발생 가능)
  - 실수를 정수로 형 변환 : 소수점 이하의 값이 소멸
  - 큰 정수를 작은 정수로 형 변환 : 작은 정수의 크기에 맞춰서 상위 바이트 소멸

```
int main(void)
{
    double num1=245;
    int num2=3.1415;
    int num3=129;
    char ch=num3;

    printf("정수 245를 실수로: %f \n", num1);
    printf("실수 3.1415를 정수로: %d \n", num2);
    printf("큰 정수 129를 작은 정수로: %d \n", ch);
    return 0;
}
```

AutoConvOne.c

실행결과

```
정수 245를 실수로: 245.000000
실수 3.1415를 정수로: 3
큰 정수 129를 작은 정수로: -127
```

37

연산 수행 시 자료형 불일치로 발생하는 자동 형 변환

```
double num1 = 5.15 + 19;
```

두 피연산자의 자료형은 일치해야 한다.
일치하지 않으면 일치하기 위해서
자동으로 형 변환이 발생

아래의 자동 형 변환 규칙을 근거로 int형 데이터 19가 double형 데이터 19.0으로 형 변환이 되어 덧셈이 진행된다.

산술연산에서의
자동 형 변환 규칙

- 바이트 크기가 큰 자료형이 우선시 된다.
- 정수형보다 실수형을 우선시 한다.
- 이는 데이터의 손실을 최소화 하기 위한 기준이다.

38

type 변환: 강제로 일으키는 수동 형 변환

ConvDiv.c

```
int main(void)
{
    int num1=3, num2=4;
    double divResult;
    divResult = num1 / num2;
    printf("나눗셈 결과: %f \n", divResult);
    return 0;
}
```

num1과 num2가 정수이기 때문에 몫만 반환이 되는 정수형 나눗셈이 진행

실행결과

```
나눗셈 결과: 0.000000
```

```
divResult = (double)num1 / num2;
```

(type)은 type형으로의 형 변환을 의미

num1이 double형으로 명시적 수동 형 변환되고, num1과 num2의 / 연산 과정에서 num2의 값은 산술 연산에서의 자동 형 변환되어 double로 되어, 실수형 나눗셈이 진행되며 divResult에는 0.75가 저장

```
int main(void)
{
    int num1 = 3;
    double num2 = 2.5 * num1;
    . . . .
}
```

⟺

```
int main(void)
{
    int num1 = 3;
    double num2 = 2.5 * (double)num1;
    . . . .
}
```

자동 형 변환이 발생하는 위치에 명시적 형 변환 표시를 해서 형 변환이 발생함을 알리는 것이 좋다!

추천하는 코드 작성 스타일

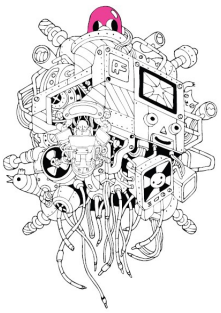
39

실습 문제 (Lab4)

- 아래의 지시에 따라 프로그램 하시오.
  1) 정수 (int)형 변수 a를 만든 후 여기에 10을 저장
  2) 정수 형 변수 b를 만든 후 여기에 3을 저장
  3) 실수 (float)형 변수 c를 만든 후 여기에 a를 b로 나눈 값을 저장, 출력
  4) 추가로, 변수 c에 제대로 된 연산 결과가 저장되도록 형 변환
  5) 정수 형 변수 d를 만들고 변수 c의 값을 저장하여 그 결과 확인
  - 아래와 같은 출력이 되도록 해야 함. 단, printf 함수 호출 시 숫자는 보이지 않도록 해야 함.

```
a의 값은 10이고, b의 값은 3입니다.
c의 값은 3.00000입니다.
강제 형 변환 후 c의 값은 3.33333입니다.
d의 값은 3입니다.
```

40



Chapter 06. printf 함수와 scanf 함수

printf 함수 이야기

printf 함수와 특수문자

```
int main(void)
{
    printf("I like programming \n");
    printf("I love puppy! \n");
    printf("I am so happy \n");
    return 0;
}
```

StringPrintf.c

실행결과

```
I like programming
I love puppy!
I am so happy
```

printf 함수는 첫 번째 인자로 전달된 문자열을 출력한다.

printf("앞집 강아지가 말했다. "멍~! 멍~!" 정말 귀엽다."); 잘못된 printf 함수 호출문

"앞집 강아지가 말했다." → 이것은 하나의 문자열이군!
멍~! 멍~! → 이건 뭐지?
"정말 귀엽다." → 이것도 하나의 문자열이고!

큰 따옴표는 문자열의 시작과 끝으로 해석이 되니, 큰 따옴표 자체의 출력을 원하는 경우에는 큰 따옴표 앞에 ₩ 문자를 붙여주기로 하자!

printf("앞집 강아지가 말했다. ₩"멍~! 멍~!₩" 정말 귀엽다."); 제대로 된 printf 함수 호출문

특수문자의 종류

특수문자	의미하는 바	
\a	경고음	
\b	백스페이스(backspace)	한 칸 (글자) 이전으로 이동
\f	폼 피드(form feed)	프린터 출력에 사용
\n	개 행(new line)	다음 줄의 가장 앞으로 이동
\r	캐리지 리턴(carriage return)	현재 출력 중인 줄의 가장 앞으로 이동
\t	수평 탭	탭 만큼 이동
\v	수직 탭	프린터 출력에 사용
\'	작은 따옴표 출력	
\"	큰 따옴표 출력	
\?	물음표 출력	
\\	역슬래쉬 출력	

₩f와 ₩v는 모니터 출력이 아닌 프린터 출력을 위해 정의된 특수문자이기 때문에 모니터의 출력에 사용하면, 이상한 문자 출력!

정수값 출력을 위한 서식문자들: %d, %u, %o, %x

```
int main(void)
{
    int myAge=12;
    printf("제 나이는 10진수로 %d살, 16진수로 %X살입니다. \n", myAge, myAge);
    return 0;
}
```

서식문자를 이용해서 출력할 문자열의 형태를 조합해 낼 수 있다.
즉, 출력의 서식을 지정할 수 있다.

FormPrintf.c

```
제 나이는 10진수로 12살, 16진수로 C살입니다.
```

실행결과

서식문자	출력 대상(자료형)	출력 형태
%d	char, short, int	부호 있는 10진수 정수
%ld	long	부호 있는 10진수 정수
%lld	long long	부호 있는 10진수 정수
%u	unsigned int	부호 없는 10진수 정수
%o	unsigned int	부호 없는 8진수 정수
%x, %X	unsigned int	부호 없는 16진수 정수
%f	float, double	10진수 방식의 부동소수점 실수
%Lf	long double	10진수 방식의 부동소수점 실수
%e, %E	float, double	e 또는 E 방식의 부동소수점 실수
%g, %G	float, double	값에 따라 %f와 %e 사이에서 선택
%c	char, short, int	값에 대응하는 문자
%s	char *	문자열
%p	void *	포인터의 주소 값

OctHex.c

```
int main(void)
{
    int num1=7, num2=13;
    printf("%o %#o \n", num1, num1);
    printf("%x %#x \n", num2, num2);
    return 0;
}
```

```
7 07
d 0xd
```

실행결과

#을 삽입하면 8진수 앞에 0, 16진수 앞에 0x가 삽입된다.

실수값 출력을 위한 서식문자들: %f, %e, %g

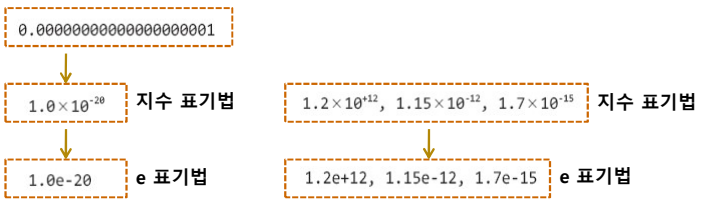
```
int main(void)
{
    printf("%f \n", 0.1234);
    printf("%e \n", 0.1234);    // e 표기법 기반의 출력
    printf("%f \n", 0.12345678);
    printf("%e \n", 0.12345678);    // e 표기법 기반의 출력
    return 0;
}
```

RealNumOutput.c

실행결과

```
0.123400
1.234000e-001
0.123457
1.234568e-001
```

컴퓨터는 지수를 표현할 수 없으므로 e 표기법으로 지수를 대신 표현한다.



%g의 실수출력과 %s의 문자열 출력

```
int main(void)
{
    double d1=1.23e-3;   // 0.00123
    double d2=1.23e-4;   // 0.000123
    double d3=1.23e-5;   // 0.0000123
    double d4=1.23e-6;   // 0.00000123

    printf("%g \n", d1);   // %f 스타일 출력
    printf("%g \n", d2);   // %f 스타일 출력
    printf("%g \n", d3);   // %e 스타일 출력
    printf("%g \n", d4);   // %e 스타일 출력
    return 0;
}
```

- %g는 실수의 형태에 따라서 %f와 %e 사이에서 적절한 형태의 출력을 진행한다.
- %g와 %G의 차이점은 e 표기법의 e를 소문자로 출력하느냐 대문자(E)로 출력하느냐에 있다.

실행결과 UsingPG.c

```
0.00123
0.000123
1.23e-005
1.23e-006
```

```
int main(void)
{
    printf("%s, %s, %s \n", "AAA", "BBB", "CCC");
    return 0;
}
```

UsingPS.c

실행결과

```
AAA, BBB, CCC
```

%s의 문자열 출력과 관련해서는 배열과 포인터 공부 후에 완벽히 이해하자!
일단은 %s의 사용법을 예제 기반으로 이해하자.

필드 폭을 지정하여 정돈된 출력 보이기

%8d
필드 폭을 8칸 확보하고, 오른쪽 정렬해서 출력을 진행한다.

%-8d
필드 폭을 8칸 확보하고, 왼쪽 정렬해서 출력을 진행한다.

%+8d
필드 폭을 8칸 확보하고, 오른쪽 정렬한 상태에서 양수는 +, 음수는 –를 붙여서 출력한다.

```
int main(void)
{
    printf("%-8s %14s %5s \n", "이  름", "전공학과", "학년");
    printf("%-8s %14s %5d \n", "김동수", "전자공학", 3);
    printf("%-8s %14s %5d \n", "이을수", "컴퓨터공학", 2);
    printf("%-8s %14s %5d \n", "한선영", "미술교육학", 4);
    return 0;
}
```

FieldWidth.c

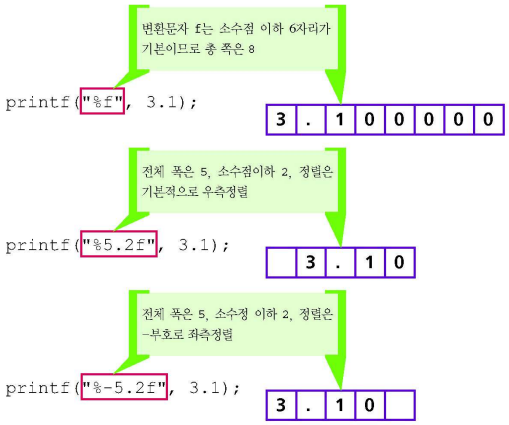
실행결과

```
이  름          전공학과     학년
김동수          전자공학        3
이을수        컴퓨터공학        2
한선영        미술교육학        4
```

서식문자 사이에 들어가는 숫자는 필드의 폭을 의미한다.
기본은 오른쪽 정렬이다. 따라서 –는 왼쪽 정렬을 의미하는 용도로 사용된다.

실수의 필드 폭

- 부동소수형을 출력하는 경우
  - 변환명세에서 필드 폭(width)을 지정하려면 %와 f 사이에 폭을 기술



실습 문제(Lab3)

- 아래의 지시에 따라 프로그램 하시오.
  1) 정수 (int)형 변수 a를 만든 후 여기에 100을 저장
  2) 작은 정수 (short)형 변수 b를 만든 후 여기에 a의 값을 저장
  3) 실수 (float)형 변수 c를 만든 후 여기에 3.54를 저장
  4) double형 변수 d를 생성과 동시에 456.789를 저장 (const 설정)
  5) sizeof 함수를 이용해서 각 변수 (방)의 크기를 출력
  6) 변수 d의 값을 임의의 다른 값으로 변경해 볼 것
  - 아래와 같은 출력이 되도록 한다. 단, 아래의 숫자들은 printf 함수 호출 시 보이지 않아야 함.

```
a의 값은 100이고, c의 값은 3.540000이고,
d의 값은 456.789000이고, b의 값은 100입니다.
a의 크기는 4, c의 크기는 4, d의 크기는 8,
b의 크기는 2 바이트 입니다.
```

50

실습 문제 (Lab4)

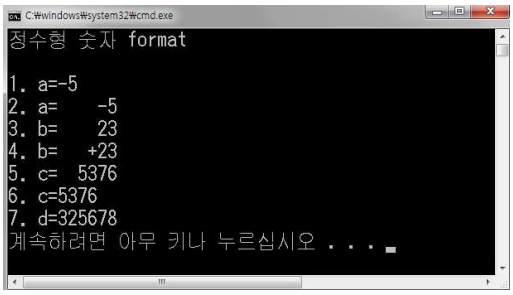
- 아래의 지시에 따라 프로그램 하시오.
  1) 정수 (int)형 변수 a를 만든 후 여기에 10을 저장
  2) 정수 형 변수 b를 만든 후 여기에 3을 저장
  3) 실수 (float)형 변수 c를 만든 후 여기에 a를 b로 나눈 값을 저장, 출력
  4) 추가로, 변수 c에 제대로 된 연산 결과가 저장되도록 형 변환
  5) 정수 형 변수 d를 만들고 변수 c의 값을 저장하여 그 결과 확인
  - 아래와 같은 출력이 되도록 해야 함. 단, printf 함수 호출 시 숫자는 보이지 않도록 해야 함.

```
a의 값은 10이고, b의 값은 3입니다.
c의 값은 3.00000입니다.
강제 형 변환 후 c의 값은 3.33333입니다.
d의 값은 3입니다.
```

51

실습 문제 (Lab5)

- 아래와 같은 출력 결과가 나오도록 프로그램을 만드시오
  - a는 정수 값 -5, b는 정수 값 23, c는 정수 값 5376, d는 정수 값 325678을 저장
  - printf 함수의 필드 폭 조정 값을 이용하여 아래와 같은 모양과 동일한 모양으로 출력
  - Printf 함수의 " "안에는 서식 문자만 사용할 것!!



실습 시간 (2019년 4월 10일)

- 예제 문제 (10개)
  - 5장: LiteralSize.c, AutoConvOne.c ConvDiv.c
  - 6장: StringPrintf.c, FormPrintf.c, OctHex.c, RealNumOutput.c, UsingPG.c, UsingPS.c, FieldWidth.c,
- 실습 문제 (1개)
  - Lab5

53

scanf 함수 이야기

정수 기반의 입력형태 정의하기

입력의 형식 어떻게 받아들일 거니?
입력의 장소 어디에 저장할까?

데이터를 입력 받는
scanf 함수에게 전달해야 할 두 가지 정보

%d 10진수 정수의 형태로 데이터를 입력 받는다.
%o 8진수 양의 정수의 형태로 데이터를 입력 받는다.
%x 16진수 양의 정수의 형태로 데이터를 입력 받는다.

서식문자의 의미는 출력을
입력으로만 변경하면
printf 함수와 유사하다.

```
int main(void)
{
    int num1, num2, num3;
    printf("세 개의 정수 입력: ");
    scanf("%d %o %x", &num1, &num2, &num3);
    printf("입력된 정수 10진수 출력: ");
    printf("%d %d %d \n", num1, num2, num3);
    return 0;
}
```

ScanfConvOne.c

실행결과

```
세 개의 정수 입력: 12 12 12
입력된 정수 10진수 출력: 12 10 18
```

실수 기반의 입력형태 정의하기

float형 데이터의 삽입을 위한 서식문자

printf 함수에서는 서식문자 %f, %e 그리고 %g의 의미가 각각 달랐다.
그러나 scanf 함수에서는 'float형 데이터를 입력 받겠다'는 동일한 의미를 담고 있다.

double형 long double형 데이터의 삽입을 위한 서식문자

%lf double %f에 l이 추가된 형태
%Lf long double %f에 L이 추가된 형태

float, double, long double의 데이터 출력 %f, %f, %Lf
float, double, long double의 데이터 입력 %f, %lf, %Lf

실수의 입력과정에서
e 표기법을 사용해도 된다.

실행결과

```
실수 입력1(e 표기법으로): 1.1e-3
입력된 실수 0.001100
실수 입력2(e 표기법으로): 0.1e+2
입력된 실수 10.000000
실수 입력3(e 표기법으로): 0.17e-4
입력된 실수 0.000017
```

ScanfConvTwo.c

```
int main(void)
{
    float num1;
    double num2;
    long double num3;
    printf("실수 입력1(e 표기법으로): ");
    scanf("%f", &num1);
    printf("입력된 실수 %f \n", num1);
    printf("실수 입력2(e 표기법으로): ");
    scanf("%lf", &num2);
    printf("입력된 실수 %f \n", num2);
    printf("실수 입력3(e 표기법으로): ");
    scanf("%Lf", &num3);
    printf("입력된 실수 %Lf \n", num3);
    return 0;
}
```

scanf 요약 정리

- 키보드로부터 정수를 입력 받아 저장하려고 하는 경우
  - int num1;
  - scanf (“%d”, &num1);
- 키보드로부터 실수를 입력 받아 저장하려고 하는 경우
  - float num2;
  - scanf (“%f”, &num2);
- 키보드로부터 큰 실수를 입력 받아 저장하려고 하는 경우
  - double num3;
  - scanf (“%lf”, &num3);
- 키보드로부터 문자 한 글자를 입력 받아 저장하려고 하는 경우
  - char num4;
  - scanf (“%c”, &num4);
- 키보드로부터 정수 여러 개를 입력 받아 저장하려고 하는 경우
  - int num5, num6;
  - scanf (“%d %d”, &num5, &num6);

실습 문제 (Lab6)

- 아래와 같은 입출력 결과가 나오도록 프로그램을 만드시오
  - a와 b는 실수 값을 저장하기 위한 방 이름이므로 우선 이들 방을 만들어야 함
  - scanf 함수를 이용하여 a의 값과 b의 값을 입력 받음
  - 입출력 결과의 형태는 다음과 같음

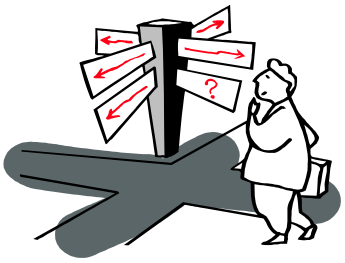
```
사칙연산 프로그램
a와 b의 값을 입력하시오.
a = 128.7
b = 23.8

연산 결과
a+b = 152.50
a-b = 104.90
a*b = 3063.06
a/b = 5.41
```

실습 시간 (2019년 4월 17일)

- 예제 문제
  - 6장: ScanfConvOne.c, ScanfConvTwo.c
- 실습 문제
  - Lab6

59



Chapter 06이 끝났습니다. 질문 있으신지요?