C 프로그래밍 1
Lecture Note #05

백윤철
ybaek@smu.ac.kr

내용

- 자료형
- sizeof 연산
- 정수 자료형
- 실수 자료형
- 문자 표현을 위한 자료형
- 상수
- 자료형의 변환

자료형은 데이터를 표현하는 방법

- 실수를 저장할 것이냐? 정수를 저장할 것이냐?
  - 값을 저장하는 방식이 실수냐 정수냐에 따라 다르기 때문에 용도를 결정해야 함
- 얼마나 큰 수를 저장할 것이냐?
  - 큰 수를 표현하기 위해서는 많은 수의 바이트가 필요함

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.."

int num;

자료형은 데이터를 표현하는 방법

- 자료형의 수는 데이터 표현 방법의 수를 의미함
- C언어가 제공하는 기본 자료형의 수가 10개라면, C언어가 제공하는 기본적인 데이터 표현방식의 수는 10개라는 뜻이 됨

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

자료형은 데이터를 표현하는 방법

- 컴파일러에 따라서 약간의 차이를 보임
  - C언어 표준에서는 자료형 별 상대적 크기를 표준화 할 뿐 구체적인 크기까지는 언급하지 않음
- 크게 정수형과 실수형으로 나눔
  - 데이터를 표현하는 방식이 달라지므로
- 정수형에도 실수형에도 두 개 이상의 기본 자료형 존재
  - 표현하고자 하는 값의 크기에 따라서 적절히 선택할 수 있도록 다수의 자료형 제공

연산자 sizeof를 이용한 바이트 크기 확인

- sizeof연산자의 피연산자로는 변수, 상수, 자료형의 이름 등이 올 수 있음
- 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));
```

연산자 sizeof를 이용한 바이트 크기 확인

```
    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;
}
```

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
```

정수의 표현 및 처리를 위한 일반적 자료형 선택

- 일반적 선택은 int임
  - CPU가 연산하기에 가장 적합한 데이터의 크기가 int형으로 정해짐 → 가장 연산이 빠른 걸로 생각하면 됨
  - 연산이 동반이 되면 int형으로 형 변환 되어서 연산 진행
  - 따라서 연산을 동반하는 변수의 정의를 위해서는 int로 정의하는 것이 바람직함
  - 실행 속도가 중요하다면 int 사용
- char형 또는 short형은 불필요한가?
  - 연산을 수반하지 않으면서 많은 수의 데이터를 저장해야 한다면, 그리고 그 데이터의 크기가 char 또는 short로 충분히 표현 가능하다면, char 또는 short 사용 → 메모리를 최소한으로 사용하려 한다면 char 또는 short 사용

정수의 표현 및 처리를 위한 일반적 자료형 선택

```
int main(void) {
    char num1 = 1, num2 = 2, result1 = 0;
    short num3 = 300, num4 = 400, result2 = 0;
    printf("size of num1 & num2: %d, %d\n",
            sizeof(num1), sizeof(num2));
    printf("size of num3 & num4: %d, %d\n",
            sizeof(num3), sizeof(num4));
    printf("sizeof char add: %d\n",
            sizeof(num1 + num2));
    printf("sizeof short add: %d\n",
            sizeof(num3 + num4));
```

정수의 표현 및 처리를 위한 일반적 자료형 선택

```
    result1 = num1 + num2;
    result2 = num3 + num4;
    printf("sizeof result1 & result2: %d, %d\n",
            sizeof(result1), sizeof(result2));
    return 0;
}
```

실수의 표현 및 처리를 위한 일반적 자료형 선택

- 실수 자료형의 선택 기준은 정밀도
  - 실수의 표현 범위는 float, double 둘 다 충분히 넓음
  - 그러나 8바이트 크기의 double이 float 보다 더 정밀하게 실수 표현 가능
- 일반적인 선택은 double
  - 컴퓨팅 환경의 발전으로 double 형 데이터의 표현 및 연산이 덜 부담스러움
  - float 형 데이터의 정밀도는 부족한 경우가 있음

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

실수의 표현 및 처리를 위한 일반적 자료형 선택

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

double형 변수의 출력 서식문자 %lf

double형 변수의 입력 서식문자 %lf

실행결과

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

unsigned를 붙여서 0과 양의 정수만 표현

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

- 정수 자료형의 이름 앞에는 unsigned를 붙일 수 있음
- unsigned가 붙으면 MSB도 데이터의 크기를 표현
- 표현하는 값의 범위가 양의 정수로 되며, 두 배 늘어남

문자의 표현을 위한 약속! 아스키(ASCII) 코드 !

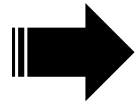
아스키 코드	아스키 코드 값
A	65
B	66
C	67
~	126

- 미국 표준 협회(ANSI)에서 제정된 '아스키(ASCII: American Standard Code for Information Interchange) 코드'

- 컴퓨터는 문자를 표현 및 저장 못함. 따라서 문자 표현을 목적으로 각 문자에 고유한 숫자를 지정
- 인간이 입력하는 문자는 해당 문자의 숫자로 변환되어 컴퓨터에 저장 및 인식되고, 컴퓨터에 저장된 숫자는 문자로 변환되어 인간의 눈에 보이게 됨

문자의 표현을 위한 약속! 아스키(ASCII) 코드 !

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



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

- 컴파일 할 때 각 문자는 해당 아스키 코드 값으로 변환
- 따라서 실제 컴퓨터에게 전달되는 데이터는 문자가 아닌 숫자임
- C프로그램 상에서 문자는 작은 따옴표로 묶어서 표현함

문자는 이렇게 표현되는 거구나!

```
int main(void) {
    char ch1 = 'A', ch2 = 65;
    char 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);
}
```

실행결과

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

문자는 이렇게 표현되는 거구나!

- 문자를 char형 변수에 저장하는 이유
- 모든 아스키 코드 문자는 1바이트로도 충분히 표현 가능
- 문자는 덧셈, 뺄셈과 같은 연산을 동반하지 않음. 단지 표현에만 사용됨
- 따라서 1바이트 크기인 char형 변수가 문자를 저장하기 위한 최적의 장소가 됨
- 문자는 int형 변수에도 저장 가능

ASCII TABLE

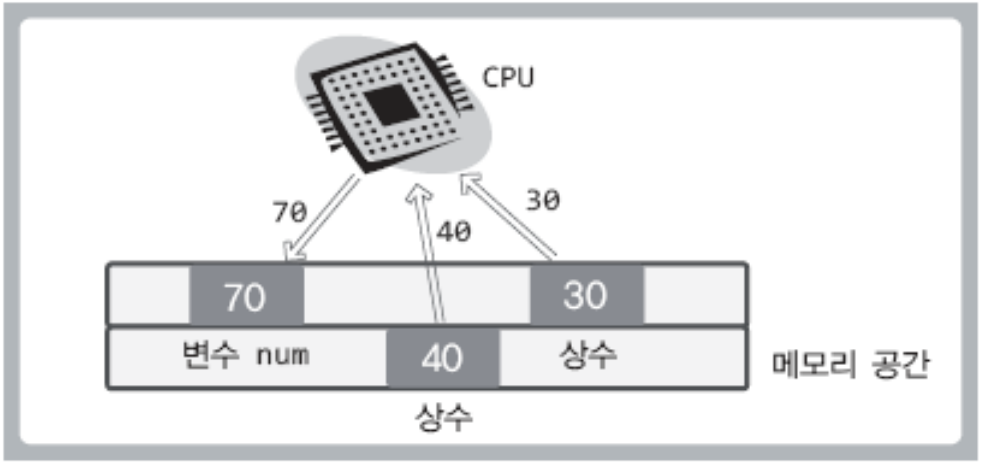
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[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	[NEGATIVE 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	[ENG OF 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	[SUBSTITUTE]	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	[DEL]

이름 없는 리터럴 상수

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

- 연산을 위해서는 30, 40과 같이 프로그램에 표현되는 숫자도 메모리 공간에 저장되어야 함
- 이렇게 저장되는 값은 이름이 존재하지 않으므로 변경이 불가능한 상수임
- 이러한 상수를 리터럴(Literal) 상수라고 함

이름 없는 리터럴 상수



메모리 공간에 저장이 되어야 CPU의 연산대상이 된다.

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

리터럴 상수의 자료형

```
int main(void) {
   printf("literal int size: %d\n", sizeof(7));
   printf("literal double size: %d\n",
          sizeof(7.14));
   printf("literal char size: %d\n",
          sizeof('A'));
   return 0;
}
```

- 정수는 기본적으로 int형으로 표현됨
- 실수는 기본적으로 double형으로 표현됨
- 문자는 기본적으로 int형으로 표현됨

접미사를 이용한 다양한 상수의 표현

```
int main(void) {
    float num1 = 5.789; /* 경고 메시지 발생 */
    float num2 = 3.24 + 5.12; /* 경고 메시지 발생 */
    return 0;
}
```

- 실수는 double 형 상수로 인식되어 데이터 손실에 대한 경고 메시지 발생

```
int main(void) {
   float num1 = 5.789f; /* 경고 메시지 발생 안함 */
   float num2 = 3.24F + 5.12F; /*대문자 F 사용 가능*/
   return 0;
}
```

접미사를 이용한 다양한 상수의 표현

정수형 상수의 표현을 위한 접미사

접미사	자료형	사용 예
U	unsigned int	unsigned n = 1025U;
L	long	long n = 2467L;
UL	unsigned long	unsigned long n = 3456UL;
LL	long long	long long n = 5768LL;
ULL	unsigned long long	unsigned long long n = 8978ULL;
F	float	float f = 3.15F
L	long double	long double f = 5.789L

실수형 상수의 표현을 위한 접미사

접미사	자료형	사용 예
F	float	float f = 3.15F
L	long double	long double f = 5.789L

이름을 지니는 심볼릭 (Symbolic) 상수: const 상수

```
int main(void) {
    const int MAX = 100; /* MAX 값 변경 불가 */
    const double PI = 3.1415; /* PI 값 변경 불가 */
    …
}
```

```
int main(void) {
    const int MAX; /* 쓰레기 값으로 초기화 됨 */
    MAX = 100; /* 값 변경 불가. 컴파일 에러 발생 */
    …
}
```

- 상수의 이름은 모두 대문자로 표시하고 둘 이상의 단어를 사용할 때에는 '_'를 이용해 구분 (MY_AGE)

대입 연산의 과정에서 발생하는 자동 형 변환

- 대입 연산자의 왼편을 기준으로 형 변환 발생

```
double num1 = 245; /* int → double 자동 형 변환 */
int num2 = 3.1415; /* double → int 자동 형 변환 */
```

```
int num3 = 129;
char ch = num3; /* int → char 자동 형 변환 */
```

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

00000000 00000000 00000000 10000001 ➡ 10000001

자동 형 변환의 방식 정리

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

자동 형 변환의 방식 정리

```
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;
}
```

실행결과

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

정수의 승격에 의한 자동 형 변환

- 일반적으로 CPU가 처리하기에 가장 적합한 크기의 정수 자료형을 int로 정의
- 따라서 int형 연산의 속도가 다른 자료형의 연산 속도에 비해 동일하거나 더 빠름
- 따라서 다음과 같은 방식의 형 변환 발생

```
int main(void) {
    short num1 = 15, num2 = 25;
    /* num1과 num2가 int형으로 형 변환 */
    /* 정수의 승격(Integral Promotion)이라고 함 */
    short num3 = num1 + num2;
    …
}
```

피연산자의 자료형 불일치로 발생하는 자동 형 변환

- 두 피연산자의 자료형은 일치해야 함. 일치하지 않으면 일치시키기 위해 자동으로 형 변환 발생

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

실수형으로 바뀜

산술연산에서의 자동 형 변환 규칙

바이트 크기가 큰 자료형이 우선시 된다.

정수형보다 실수형을 우선시 한다.

이는 데이터의 손실을 최소화 하기 위한 기준이다.

명시적 형 변환: 강제로 형 변환

```
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
```

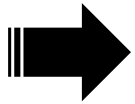
```
divResult = (double) num1 / num2;
```

(type)은 type형으로의 형 변환을 의미한다.

num1이 double형으로 명시적 형 변환 그리고 num1과 num2의 / 연산 과정에서의 산술적 자동 형 변환! 그 결과 실수형 나눗셈이 진행되어 divResult에는 0.75가 저장된다.

명시적 형 변환: 강제로 형 변환

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



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

추천하는 코드 작성 스타일

- 자동 형 변환이 발생하는 위치에 명시적 형 변환 표시를 해서 형 변환이 발생함을 알리는 것이 좋음

정리

- 자료형
- sizeof 연산
- 정수 자료형
  - int, char, short, long
- 실수 자료형
  - double, float
- 문자 표현을 위한 자료형
  - char
- 상수
- 자료형의 변환