학번 : _____ 성명 : _____.

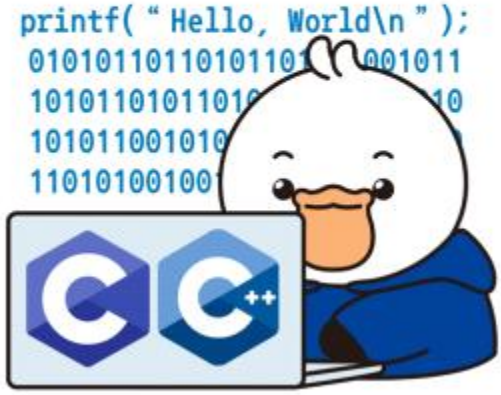
• 본 강의자료는 과학기술정보통신부 및 정보통신기획평가원에서 지원하는『소프트웨어중심대학』사업의 결과물입니다.

SW 대구가톨릭대학교 소프트웨어중심대학사업단

대구가톨릭대학교 컴퓨터소프트웨어학부 School of Computer Software, Daegu Catholic University

Part 2. 데이터와 데이터 타입

목차



데이터와 데이터 타입

- 데이터(data)
  - ✓ 일반적인 상황에서 자료라는 의미로 주로 사용
  - ✓ 프로그래밍에서는 프로그램에 사용하거나 저장하기 위한 값을 의미
- 데이터 타입(data type)
  - ✓ 자료형이라고도 하며 프로그래밍 언어에서 실수, 정수, 참, 거짓, 문자 등의 다양한 데이터 분류를 위해 사용
  - ✓ C언어에서는 자료형을 크게 [ ], [ ], [ ] 으로 구분

정수형 실수형 문자형

2.1 정수형 데이터 타입

char, int, long, signed, unsigned

- 정수 값 저장을 위한 데이터 타입
  - ✓ C에서는 [ ], [ ], [ ], [ ], [ ] 등의 데이터 타입을 제공
  - ✓ 각각의 프로그램이 동작하는 환경에서 [ ]로 [ ] 후 실행 (기계어 · 컴퓨터)
    - ➢ 실행 환경 CPU의 데이터 처리 방식에 따라 데이터 타입의 크기 결정
  - ✓ 저장할 수 있는 정수 범위에 따라 다른 데이터 타입 제공
    - ➢ **0, 양수** 저장 -> [ ] unsigned
    - ➢ **음수, 0, 양수** 저장 -> [ ] signed

2.1 정수형 데이터 타입

- char (바이트 (byte))
  - ✓ 1 [ ] 크기의 정수를 저장할 수 있음
  - ✓ 문자를 저장하기 위해 주로 사용하지만 기본적으로 정수형 데이터 타입
- int
  - ✓ 일반적으로 [ ] 크기의 정수를 저장 (4 바이트)
    - ➢CPU 동작 비트 수에 따라 다른 크기로 사용되므로 사전 확인 필요
  - ✓ C언어에서 기본적으로 사용하는 정수형 데이터 타입

2^{32}

2.1 정수형 데이터 타입

✓ short

➢ [ ]크기의 정수를 저장

➢ char보다 크고, int보다 작은 정수형 데이터 저장

✓ long

➢크기가 변하는 int와 달리 늘 [ ]크기 정수를 저장

✓ long long

➢ [ ]크기의 정수를 저장

➢ int, long보다 더 큰 정수 저장을 위해 사용

2.1 정수형 데이터 타입

✓ signed

➢ [ ], [ ], [ ]를 저장할 때 사용하는 지정자

➢ char, int, short, long, long long의 경우 기본적으로 signed가 생략

✓unsigned

➢ 0과 양수만 저장할 때 사용하는 지정자

➢ 표현 범위가 변하는 것이기 때문에 자료형의 [ ]는 불변

2.1 정수형 데이터 타입의 크기와 표현 범위

▪ 정수형 데이터 타입의 크기와 표현 범위

✓데이터 타입의 범위에 벗어난 수를 저장 시 데이터 타입의 최대 또는 최솟값이 저장

자료형		크기 (바이트)	범위	비고
char	signed char	1바이트	-128~127 (-2^7~2^7-1)	signed 생략가능
unsigned char		1바이트	0~255 (0~2^8-1)	
short short int	signed short signed short int	2바이트	-32,768~32,767 (-2^{15}~2^{15}-1)	int 및 signed 생략가능
unsigned short unsigned short int		2바이트	0~65,535 (0~2^{16}-1)	
int	signed int	4바이트	-2,147,483,648~2,147,483,647 (-2^{31}~2^{31}-1)	
unsigned unsigned int		4바이트	0~4,294,967,295 (0~2^{32}-1)	int 및 signed 생략가능
long long int	signed long signed long int	4바이트	-2,147,483,648~2,147,483,647 (-2^{31}~2^{31}-1)	
unsigned long unsigned long int		4바이트	0~4,294,967,295 (0~2^{32}-1)	
long long long long int	signed long long signed long long int	8바이트	-9,223,372,036,854,775,808~ 9,223,372,036,854,775,807 (-2^{63}~2^{63}-1)	
unsigned long long unsigned long long int		8바이트	0~18,446,744,073,709,551,615 (0~2^{64}-1)	

2.1 정수형 데이터 타입

▪ 데이터 타입의 크기 바이트(byte) · 비트(bit)

✓ [ ] 단위 또는 [ ] 단위로 표현

✓ [ ] 비트

➢ 컴퓨터가 처리할 수 있는 데이터의 최소 단위

➢ 하나의 비트에 0 또는 1의 값을 저장

✓ [ ] 바이트

➢ 8개의 비트로 구성된 데이터 단위

운영체제	CPU	크기 (바이트, 비트)
윈도우 계열	x86	4byte, 32bit
	x64	4byte, 32bit
리눅스 계열	x86	4byte, 32bit
	x64	8byte, 64bit
맥 계열	x86	4byte, 32bit
	x64	8byte, 64bit

<운영체제 별 정수형 데이터 타입의 크기>

2.1 정수형 데이터 타입

소스 2-1

```
1  #include <stdio.h>
2  int main()
3  {
4          char ch = -20;
5          short sh = 5400;
6          int i = -123456789;
7          long l = 123456789;
8          long long ll = -123456789012;
9          // char, short, int 출력시 %d로 출력, long은 %ld, long long은 %lld로 출력
10         printf("%d %d %d %ld %lld\n", ch, sh, i, l, ll);
11         return 0;
12 }
```

출력 예

```
-20 5400 -123456789 123456789 -123456789012
```


2.1 정수형 데이터 타입

소스 2-2

```
1  /* 부호 없는 정수의 산술연산이 오버플로우를 일으키지 않는지 확인 */
2  #include <stdio.h>
3  #include <limits.h> // 데이터 타입 별 최댓값과 최솟값이 정의 된 헤더파일
4  int main()
5  {
6          unsigned int u1 = 10;
7          unsigned u2 = 4294967295;
8          printf("UINT_MAX는 %u입니다.\n", UINT_MAX);
9          printf("u1 : %u\n", u1);
10         printf("u2 : %u\n", u2);
11         printf("u1 + u2 = %u\n", u1 + u2);
12         return 0;
13 }
```

출력 예

```
UNIT_MAX는 4294967295입니다.
u1 : 10
u2 : 4294967295
u1 + u2 = 9
```

2.1 정수형 데이터 타입

소스 2-3

```
1  /* 2개의 변수를 초기화하고 출력 */
2  #include <stdio.h>
3  int main()
4  {
5          int i1 = 3.14;        /* i1는 int형 변수(3.14로 초기화)*/
6          int i2 = -4.5;        /* i2는 int형 변수(-4.5로 초기화) */
7          printf("i1의 값은 %d입니다.\n", i1); /* vx의 값을 출력 */
8          printf("i2의 값은 %d입니다.\n", i2); /* vy의 값을 출력 */
9          return 0;
10 }
```

출력 예
```
i1의 값은 3입니다.
i2의 값은 -4입니다.
```


2.2 실수형 데이터 타입

- 실수 값 저장을 위한 데이터 타입
  - ✓ [ ] - 단정밀도(single precision), 4바이트 크기
  - ✓ [ ] - 배정밀도(double precision), 8바이트 크기
  - ✓ C에서 [ ]는 소수점 7자리까지, [ ]은 16자리까지 표현 가능

float

double

float / double

자료형	크기				유효소수점 이하 자리수
	전체크기	부호	지수	가수	
[ ] double	4바이트	1비트	8비트	23비트	7
	8바이트	1비트	11비트	52비트	16
[ ] double	8바이트	1비트	11비트	52비트	16

< 실수형 데이터 타입 크기와 표현 범위 >

운영체제	CPU(플랫폼)	바이트 크기	비트 크기
Windows	x86 (32비트)	8	64
	x64 (64비트)	8	64
리눅스	x86 (32비트)	12	96
	x64 (64비트)	16	128
OS X	x86 (32비트)	16	128
	x64 (64비트)	16	128

< 운영체제 별 실수형 데이터 타입의 크기 비교 >

2.2 실수형 데이터 타입

- 2진수를 통한 실수 값 표현 방식
  - ✓ [ ] 방식과 [ ] 방식 존재
  - ✓ C언어는 국제표준 IEEE 754를 따르는 부동소수점 방식 사용
  - ✓ 따라서, double, float의 경우 의도하지 않은 값이 저장될 수 있음
    - ➢소스 2-4의 코드를 실행하여 확인 가능

부동소수점 / 고정소수점

2.2 실수형 데이터 타입

소스 2-4

```
1  /* 읽어 들인 실숫값을 그대로 출력 */
2  #include <stdio.h>
3  int main()
4  {
5          double d = 3.14;
6          float f = 3.14f;
7          printf("double : %lf %.10lf\n", d, d);
8          printf("float : %f %.10f\n", f, f);
9          return 0;
10 }
```

출력 예

```
double : 3.140000 3.1400000000
float : 3.140000 3.1400001049
```

2.2 실수형 데이터 타입

소스 2-5

```
1  /* 읽어 들인 실숫값 2개의 비율을 백분율(실수)로 출력 */
2  #include <stdio.h>
3  int main(void)
4  {
5          double numerator = 53;
6          double denominator = 128;
7          printf("분자는 분모의 %f%%입니다.\n", numerator / denominator * 100);
8          printf("분자는 분모의 %.2f%%입니다.\n", numerator / denominator * 100);
9          return 0;
10 }
```

출력 예

```
분자는 분모의 41.406250%입니다.
분자는 분모의 41.41%입니다.
```


2.3 문자형 데이터 타입

- char
  - ✓ [ ]의 약자로 문자를 저장하기 위한 데이터 타입
  - ✓ 아스키 문자에 해당하는 정수 코드값 저장
- [ ](American Standard Code for Information Interchange)
  - ✓ 영문 알파벳을 사용하는 문자 인코딩 방식
  - ✓ 총 [ ]가지의 문자를 표현할 수 있음
    - ➢ 알파벳 대소문자 52개, 숫자 10개, 특수문자 32개, 공백문자 1개, 제어문자 33개
  - ✓ 한글은 아스키로 표현할 수 없음
    - ➢ char 데이터 타입에 저장 불가

character 약자

ASCII

128

참고

- 아스키 코드표
  - ✓ 제어문자
  - ✓ https://ko.wikipedia.org/wiki/ASCII

이진법	팔진법	십진법	십육진법	약자	설명	한국어 설명
000 0000	0	0	0	NUL	Null Character	NULL
000 0001	1	1	1	SOH	Start of Header	헤더 시작
000 0010	2	2	2	STX	Start of Text	본문 시작, 헤더 종료
000 0011	3	3	3	ETX	End of Text	본문 종료
000 0100	4	4	4	EOT	End of Transmission	전송 종료, 데이터 링크 초기화
000 0101	5	5	5	ENQ	Enquiry	응답 요구
000 0110	6	6	6	ACK	Acknowledgment	긍정응답
000 0111	7	7	7	BEL	Bell	경고음
000 1000	10	8	8	BS	Backspace	백스페이스
000 1001	11	9	9	HT	Horizontal Tab	수평 탭
000 1010	12	10	0A	LF	Line feed	개행
000 1011	13	11	0B	VT	Vertical Tab	수직 탭
000 1100	14	12	0C	FF	Form feed	다음 페이지
000 1101	15	13	0D	CR	Carriage return	복귀
000 1110	16	14	0E	SO	Shift Out	확장문자 시작
000 1111	17	15	0F	SI	Shift In	확장문자 종료
001 0000	20	16	10	DLE	Data Link Escape	전송 제어 확장
001 0001	21	17	11	DC1	Device Control 1	장치 제어 1
001 0010	22	18	12	DC2	Device Control 2	장치 제어 2
001 0011	23	19	13	DC3	Device Control 3	장치 제어 3
001 0100	24	20	14	DC4	Device Control 4	장치 제어 4
001 0101	25	21	15	NAK	Negative Acknowledgement	부정응답
001 0110	26	22	16	SYN	Synchronous idle	동기
001 0111	27	23	17	ETB	End of Transmission Block	전송블록 종료
001 1000	30	24	18	CAN	Cancel	무시
001 1001	31	25	19	EM	End of Medium	매체 종료
001 1010	32	26	1A	SUB	Substitute	치환
001 1011	33	27	1B	ESC	Escape	제어기능 추가
001 1100	34	28	1C	FS	File Separator	파일경계 할당
001 1101	35	29	1D	GS	Group Separator	레코드 그룹경계 할당
001 1110	36	30	1E	RS	Record Separator	레코드 경계 할당
001 1111	37	31	1F	US	Unit Separator	장치 경계 할당
111 1111	177	127	7F	DEL	Delete	삭제

참고

- 아스키 코드표
  - ✓ 출력 가능 문자
  - ✓ https://ko.wikipedia.org/wiki/ASCII

이진법	팔진법	십진법	십육진법	모양	이진법	팔진법	십진법	십육진법	모양	이진법	팔진법	십진법	십육진법	모양
100000	40	32	20	␣	1000000	100	64	40	@	1100000	140	96	60	`
100001	41	33	21	!	1000001	101	65	41	A	1100001	141	97	61	a
100010	42	34	22	"	1000010	102	66	42	B	1100010	142	98	62	b
100011	43	35	23	#	1000011	103	67	43	C	1100011	143	99	63	c
100100	44	36	24	$	1000100	104	68	44	D	1100100	144	100	64	d
100101	45	37	25	%	1000101	105	69	45	E	1100101	145	101	65	e
100110	46	38	26	&	1000110	106	70	46	F	1100110	146	102	66	f
100111	47	39	27	'	1000111	107	71	47	G	1100111	147	103	67	g
101000	50	40	28	(	1001000	110	72	48	H	1101000	150	104	68	h
101001	51	41	29	)	1001001	111	73	49	I	1101001	151	105	69	i
101010	52	42	2A	*	1001010	112	74	4A	J	1101010	152	106	6A	j
101011	53	43	2B	+	1001011	113	75	4B	K	1101011	153	107	6B	k
101100	54	44	2C	,	1001100	114	76	4C	L	1101100	154	108	6C	l
101101	55	45	2D	-	1001101	115	77	4D	M	1101101	155	109	6D	m
101110	56	46	2E	.	1001110	116	78	4E	N	1101110	156	110	6E	n
101111	57	47	2F	/	1001111	117	79	4F	O	1101111	157	111	6F	o
110000	60	48	30	0	1010000	120	80	50	P	1110000	160	112	70	p
110001	61	49	31	1	1010001	121	81	51	Q	1110001	161	113	71	q
110010	62	50	32	2	1010010	122	82	52	R	1110010	162	114	72	r
110011	63	51	33	3	1010011	123	83	53	S	1110011	163	115	73	s
110100	64	52	34	4	1010100	124	84	54	T	1110100	164	116	74	t
110101	65	53	35	5	1010101	125	85	55	U	1110101	165	117	75	u
110110	66	54	36	6	1010110	126	86	56	V	1110110	166	118	76	v
110111	67	55	37	7	1010111	127	87	57	W	1110111	167	119	77	w
111000	70	56	38	8	1011000	130	88	58	X	1111000	170	120	78	x
111001	71	57	39	9	1011001	131	89	59	Y	1111001	171	121	79	y
111010	72	58	3A	:	1011010	132	90	5A	Z	1111010	172	122	7A	z
111011	73	59	3B	;	1011011	133	91	5B	[	1111011	173	123	7B	{
111100	74	60	3C	<	1011100	134	92	5C	₩	1111100	174	124	7C	|
111101	75	61	3D	=	1011101	135	93	5D	]	1111101	175	125	7D	}
111110	76	62	3E	>	1011110	136	94	5E	^	1111110	176	126	7E	~
111111	77	63	3F	?	1011111	137	95	5F	_					

2.3 문자형 데이터 타입

소스 2-6

```
1  /* 각 과목의 학점을 A, B, C, D, F로 출력 */
2  #include <stdio.h>
3  int main(void)
4  {
5          char kor ='B';
6          char eng ='F';
7          char math ='C';
8          char sci ='D';
9          char prog ='A';
10         printf("한국어 성적은 %c입니다.\n", kor);
11         printf("영어 성적은 %c입니다.\n", eng);
12         printf("수학 성적은 %c입니다.\n", math);
13         printf("과학 성적은 %c입니다.\n", sci);
14         printf("프로그래밍 성적은 %c입니다.\n", prog);
15         return 0;
16 }
```

출력 예
한국어 성적은 B입니다. 영어 성적은 F입니다. 수학 성적은 C입니다. 과학 성적은 D입니다. 프로그래밍 성적은 A입니다.

2.3 문자형 데이터 타입

소스 2-7

```
1  /* char형에 한글 한글자를 입력하는 경우 발생하는 에러 */
2  #include <stdio.h>
3  int main(void)
4  {
5          char you ='너';
6          printf("you는 %c입니다.\n", you);
7          return 0;
8  }
```

출력 예

```
main.c:5:13: warning: multi-character character constant [-Wmultichar]
main.c:5:13: warning: overflow in implicit constant conversion [-Woverflow]
you는 ◆입니다.
```

2.3 문자형 데이터 타입

소스 2-8

```
1  /* 정수형과 문자형으로 사용하는 char형 예제 */
2  #include <stdio.h>
3  int main() {
4          char ch1 =65;
5          char ch2 ='a';
6          printf("%c의 아스키 값은 %d이다.\n", ch1, ch1);
7          printf("%c의 아스키 값은 %d이다.\n", ch1+1, ch1+1);
8          printf("%c의 아스키 값은 %d이다.\n", ch2, ch2);
9          printf("%c의 아스키 값은 %d이다.\n", ch2+2, ch2+2);
10         return 0;
11 }
```

출력 예

```
A의 아스키 값은 65이다.
B의 아스키 값은 66이다.
a의 아스키 값은 97이다.
c의 아스키 값은 99이다.
```


2.4 기본 데이터 타입 활용

소스 2-9

```
1  /* 다양한 자료형 활용 예제 프로그램 */
2  #include <stdio.h>
3  int main()
4  {
5          char schoolYear;
6          char semester;
7          int credit;
8          double score;
9          char grade;
10         schoolYear =1;
11         semester =2;
12         credit =15;
13         score =4.1;
14         grade ='A';
```

2.4 기본 데이터 타입 활용

소스 2-9

```
15          printf("## %s 학생의 %d학년 %d학기 성적표 ##\n", "김코딩",
16                  schoolYear, semester);
17          printf("소속 : %s \n", "소프트웨어학과");
18          printf("이수학점 : %d\n", credit);
19          printf("평점 : %.1f\n", score);
20          printf("학점 : %d\n", grade);
21          return 0;
22  }
```

출력 예

```
## 김코딩 학생의 1학년 2학기 성적표 ##
소속 : 소프트웨어학과
이수학점 : 15
평점 : 4.1
학점 : A
```


2.5 변수

메모리

- 컴퓨터는 모든 데이터를 메인 [ ]에 저장
  - ✓ 프로그램 실행 시 프로그램 저장 위치에서 실행 파일을 읽어 [ ]로 복사 (메모리)
  - ✓ 프로그램에서 사용되는 데이터 또한 마찬가지로 [ ]에 저장 (메모리)
  - ✓ 프로그램에서 선언된 변수들도 마찬가지로 [ ]에서 공간을 할당 받음 (메모리)
    - ➢ [ ]는 [ ](address)를 가지고 있으며 이 주소를 통해 각 변수에서 할당 받은 공간으로 접근 가능 (메모리 / 주소)
  - ✓ C언어에서는 숫자 표현을 위해 정수형과 실수형 데이터 타입을 제공
    - ➢ 생성된 변수들은 CPU의 [ ](산술연산장치)에서 처리한다 (ALU)

2.5 변수

- 변수 선언
  - ✓ 변수 선언을 위해서는 [ ]과 [ ]을 아래와 같이 작성

자료형 · 변수이름

```
int   num1;
```

  - ✓ `int num1;` 코드를 실행하면, 정수형 데이터를 저장을 위해 `int`형 데이터 타입으로 메모리에서 공간을 할당 받아 해당 주소를 `num1` 변수와 연결
  - ✓ 변수의 이름은 자유롭게 작성할 수 있으나, 아래의 규칙을 준수하여 작성
    - ➢ 알파벳과 [ ]를 사용하여 변수를 작성할 수 있다. 숫자
    - ➢ 알파벳의 [ ]를 구분하여 작성한다. 대소문자
    - ➢ _ 를 제외한 특수문자는 사용할 수 없다.
    - ➢ 영어 알파벳, _ 를 제외한 문자로 시작할 수 없다.
    - ➢ C언어의 [ ](예약어)는 사용할 수 없다. 키워드

2.5 변수

- 예약어
  - ✓ 키워드라고도 하며, 특정 목적이나 용도로 사용하기 위해 사전에 정의해 놓은 단어

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
inline	int	long	register
restrict	return	short	signed
sizeof	static	struct	switch
typedef	unio	unsigned	void
volatile	while	_Alignas	_Alignof
_Atomic	_Bool	_Complex	_Generic
_Imaginary	_Noreturn	_Static_assert	_Thread_local

<C언어에서 정의한 키워드 목록>

2.5 변수

- 좋은 변수 작성

  ✓ 코드 작성자, 읽는 사람 모두 변수의 □ 및 □을 이해할 수 있어야 한다!

  ✓ 코드 컨벤션 (Code Convention)을 지키자.

사용할 수 없는 변수명		사용 가능하지만 권장하지 않는 변수명		바람직한 변수명	
1bun	#no	PI	NAME	sum	userAge
변수1	my score	xxx	qwerty	average	my_score
이름	$1	AAA	ii	name	updateDate
합계	do	___a	aaa	index	movieName
(total)	if	mynameis	a_b_c	myNameIs	my_name

< 변수명의 작성 예시 >

2.5 변수

- 변수에 대한 값 할당 (Assignment)

  ✓ 변수에 값을 저장하는 경우 대입 연산자 □를 이용한다

  ➢ 좌측에는 사용할 변수를, 우측에는 저장할 값을 작성
  또한 우변에는 변수, 상수, 연산식, 함수 등을 자유롭게 작성할 수 있다

  ➢ 아래 코드는 num1 변수에 정수형 값 10을 저장한다는 의미이다

num1 □ 10;

할당

2.5 변수

소스 2-10

```
1  /* 변수에 정숫값을 대입하고 출력 */
2  #include <stdio.h>
3  int main(void)
4  {
5          int x; /* x는 int형 변수 */
6          x =20; /* x에 20를 대입 */
7          printf("x의 값은 %d입니다.\n", x); /* x의 값을 출력 */
8          return 0;
9  }
```

출력 예: x의 값은 20입니다.

2.5 변수

소스 2-11

```
1  /* 변수에 정숫값을 초기화하고 출력 */
2  #include <stdio.h>
3  int main(void)
4  {
5          int x =20; /* x는 int형 변수(20으로 초기화) */
6          printf("x의 값은 %d입니다.\n", x); /* x의 값을 출력 */
7          return 0;
8  }
```

출력 예: x의 값은 20입니다.

2.6 상수

- 상수
  - ✓ 변하지 않는 고정된 수를 뜻함
    - ➢ 선언 이후 값을 변경할 수 있는 변수와 달리, 상수는 값 변경이 [ ] 불가능
  - ✓ [ ]상수와 [ ]상수, [ ]상수, [ ]상수로 구분
    - ➢ 매크로와 콘스트, 열거형 상수를 묶어 [ ]상수라고 함

리터럴 · 콘스트 · 매크로 · 열거형

심볼릭

2.6 상수

- 리터럴(literal) 상수
  - ✓ 자연상태에서 존재하는 상수를 뜻함
    - ➢ 일반적으로 사용하는 정수, 실수 표현방식과 동일하게 사용
      - 10은 [ ]형 리터럴 상수, 3.14는 [ ]형 리터럴 상수이다
        101은 [ ]형 리터럴 상수이고, 3.14f는 [ ]형 리터럴 상수이다
      - 기본 타입인 int, double은 상수에 다른 글자를 추가하지 않는다
    - ➢ 정수형 리터럴 상수를 통해 8진수, 16진수와 같은 형태 표현 또한 가능
      - 8진수는 0123, 16진수는 0x123, 과 같이 숫자 앞에 0, 0x를 붙여서 표현

int / double

long / float

2.6 상수

- 리터럴(literal) 상수
  - ✓ 문자형 데이터 타입 표현
    - ➢ char형 상수는 'C'와 같이 [ ]를 사용하여 표현
    - ➢ 문자열은 "C Programming"과 같이 [ ]를 사용하여 표현
      - C에서는 문자열 처리를 위한 별도의 데이터 타입을 제공하지 않음
      - char형 데이터 타입의 배열 또는 포인터를 통해 문자열 저장 (Part. 10 참조)

2.6 상수

- 콘스트([ ]) 상수
  - ✓ 변수 앞에 키워드 [ ]를 추가하여 상수 정의
    - ➢ 상수 정의 또한 일반적인 식별자 생성 규칙을 따라 생성해야 함
    - ➢ 일반적으로 상수는 알파벳 대문자, _ 만을 사용하여 정의
    - ➢ 콘스트 상수는 선언과 동시에 초기화 필수

```
[ ] int TEN = 10 ;
```

<콘스트 상수의 선언>

2.6 상수

소스 2-12

```
1  #include <stdio.h>
2  int main()
3  {
4          const float RATE =0.5f;
5          const int CREDIT =16;
6          const char FOURTH_GRADE ='D';
7          printf("%d %f %c\n", RATE, CREDIT, FOURTH_GRADE);
8          return 0;
9  }
```

출력 예

```
0.500000 16 D
```


2.6 상수

소스 2-13

```
1  #include <stdio.h>
2  int main()
3  {
4          const int TEN =10;
5          TEN = 27; // 상수에 값을 할당하면 컴파일 에러 발생
6          printf("%d\n", TEN);
7          return 0;
8  }
```

출력 예

```
main.c: In function 'main':
main.c:5:10: error: assignment of read-only variable 'TEN'
TEN = 27; // 상수에 값을 할당하면 컴파일 에러 발생
      ^
```

2.6 상수

소스 2-14

```
1 #include <stdio.h>
2 int main()
3 {
4         const double X =10.67, Y =2.37;
5         double z;
6         z = X + Y;
7         printf("%f", z);
8         return 0;
9 }
```

출력 예

```
13.040000
```


2.6 상수

소스 2-15

```
1 #include <stdio.h>
2 #define RADIUS 100
3 #define PI 3.141592
4 int main() {
5         double circumference;
6         circumference =2 * RADIUS * PI;
7         printf("%.2f", circumference);
8         return 0;
9 }
```

출력 예

```
628.32
```

2.7 Q&A

Q 5. 어떤 타입의 정수를 쓸 것인지 어떻게 결정하지? (1/4)

A. 큰 값 (32,767 이상이거나 -32,767 이하)이 필요한 경우, long을 쓰기 바란다. 차지하는 공간이 매우 중요하다면 (큰 배열이나 많은 구조체를 쓸 경우), short를 쓰기 바란다. 이런 경우가 아니라면 int를 쓰는 것이 가장 좋다. 오버플로우 문제가 중요시되고 음수가 필요하지 않는 경우라면, 또는 비트를 다룰 때 부호 확장에서 문제가 발생하는 것을 원치 않는다면 적절한 크기를 가진 형의 unsigned 형을 쓰는 것도 좋다.
(하지만 수식에서 signed 형과 unsigned 형을 섞어 쓰는 것은 좋지않다.)

2.7 Q&A

Q 5. 어떤 타입의 정수를 쓸 것인지 어떻게 결정하지? (2/4)

A. 문자 타입도 (특히 unsigned char) 작은 정수타입으로 쓰일 수 있지만,예상치 못한 부호 확장이나 코드의 크기를 증가시킬 수 있기 때문에 바람직하지 않다. 비슷한 크기/빠르기 문제가 float와 double에서 발생할 수 있다. 변수의 주소가 필요하고 어떤 특별한 타입이 필요한 경우라면 위의 규칙들은 모두 적용되지 않다. C 언어에서 각각의 type들이 정확한 크기를 가지도록 정의되어 있다고 착각하기 쉬운데, 사실은 그렇지 않다. C 언어에서 정의하고 있는 것은 다음과 같다:

- char type은 127 이상을 저장할 수 있다;
- short int type과 int type은 32,767까지 저장할 수 있다;
- long int는 2,147,483,647까지 저장할 수 있다;
- 따라서 다음 규칙을 적용할 수 있다:
- sizeof(char) <= sizeof(short) <= sizeof(int) <=sizeof(long) <= sizeof(long long)

2.7 Q&A

Q 5. 어떤 타입의 정수를 쓸 것인지 어떻게 결정하지? (3/4)

A. 이 규칙은 char가 적어도 8bit가 되어야 한다는 것과, short int와 int는 적어도 16 bit여야 한다는 것과, long int는 적어도 32 bit가 되어야 한다는 것을 뜻한다. (각 type의 signed와 unsigned는 같은 크기를 가진다고 보장되어 있다)
ANSI C에서 특정 machine에서 각각 type의 최소값과 최대값은 <limits.h>에 정의되어 있으며, 요약하면 다음과 같다:

Base type	Min. size (bits)	Min. value (signed)	Max. value (signed)	Max. value (unsigned)
char	8	-127	127	255
short	16	-32,767	32,767	65,535
int	16	-32,767	32,767	65,535
long	32	-2,147,483,647	2,147,483,647	4,294,967,295

2.7 Q&A

Q 5. 어떤 타입의 정수를 쓸 것인지 어떻게 결정하지? (4/4)

A. 이 표는 표준이 보장하는 최소값들을 보여준다. 많은 implementation이 이보다 더 큰 값을 제공하지만, portable한 프로그램을 만들고 싶다면 이러한 것에 의존해서는 안된다.
어떠한 이유에서 정확한 크기가 필요한 경우라면 이런 경우로는 외부 저장배치(externally-imposed stroage layout)가 필요한 경우를 들 수 있다. 적절한 typedef를 쓰시기 바란다.

2.7 Q&A

Q 6. 64-bit를 지원하는 컴퓨터에서 64-bit 타입을 쓸 수 있는 방법이 있나?

A. 골치 아픈 문제이다. 다음과 같이 세 가지 방식으로 이 문제를 생각할 수 있다.

- 현존하는 16, 32-bit 시스템에서 세 정수 타입에서 (short, int, long) 두 타입이 일반적으로 같은 크기를 가진다. 64-bit 시스템에서는 세 타입이 모두 다른 크기를 가질 수 있다. 따라서 어떤 vendor들은 64-bit long int를 제공한다.
- 아쉽게도, 현존하는 많은 코드들이 int와 long이 같다고 예상하고 쓰여져 있거나, 둘 중 하나가 32bit라고 가정한 상태에서 쓰여져 있다. 이러한 코드를 그대로 유지시킬 수 있게 어떤 vendor들은 새로운, 비표준인 64-bit long long (__longlong, 또는 __very long등) 타입을 지원한다.
- 64-bit 시스템에서 int가 64bit이어야 한다는 논의도 있었다. 왜냐하면 보통 int는 시스템이 표현하는 가장 자연스러운 word 크기를 가지기 때문이다.

따라서, 64-bit 타입을 쓰고자 하는 개발자는 적절한 typedef를 써서 코드를 작성해야 한다. (또 이식성이 높은 코드를 만들기 위해서, 16, 32-bit 시스템을 위해 64-bit를 일일히 수동으로 처리할 수 있는 코드도 만들어야 할 것 이다.) Vendor들은 또, "정확히 64 bit인 타입"보다 (현재 C 표준에 존재하지 않는) "적어도 64 bit 이상인 타입"을 소개해야 한다.

2.7 Q&A

Q 7. 다음 코드는 왜 동작하지 않는가?

```
long int n =123456;
printf("%d\n", n);
```

A. long int를 출력할 때는, 반드시 l (L의 소문자) modier를 printf 포맷에 써야 (예를 들어 %ld) 합니다. 왜냐하면 printf()는 여러분이 전달한 인자의 (여기서는 변수 n의 값) 타입을 알 수 없기 때문입니다. 그러므로 반드시 올바른 포맷 (format specier)을 써야 됩니다.

2.7 Q&A

Q 8. 문자에 해당하는 수치 (ASCII 또는 다른 문자 set code) 값을 어떻게 얻을 수 있죠? 또 그 반대로 수치 값에서 그 값에 해당하는 문자를 어떻게 얻을 수 있죠?

A. C 언어에서 문자는 (컴퓨터의 문자 셋의) 문자 코드 번호로 작은 정수로서 표현됩니다. 따라서 질문한 것과 같은 변환이 필요없습니다; 즉 문자를 가지고 있다면, 그 자체가 값이 됩니다. 예를 들어, 다음 코드는:

```
int c1 ='A', c2 =65;
printf("%c %d %c %d\n", c1, c1, c2, c2);
```

ASCII를 쓰는 시스템에서, 다음과 같은 출력을 만들어 냅니다 : A 65 A 65

2.7 Q&A

Q 9. float 타입의 변수에 3.1을 넣고 printf로 출력해보니 3.09999999가 나온다. 왜 그럴까?

A. 대부분의 컴퓨터는 실수를 표현할 때, 내부적으로 2 진수로 기록한다. 2 진수에서는 0.1은 무리수로 (0.0001100110011. . .) 표현된다. 따라서 3.1과 같은 수는 (10 진수로 볼 때에는 유리수이지만) 2 진수에서는 유한한 수치로 표현될 수 없다. 여러분의 시스템에서 10 진수를 2 진수로 변환하는 루틴이 어떻게 작성되었느냐에 따라 다르겠지만, (특히 정밀도가 낮은 float을 쓸 경우) 대입한 수치와 출력 결과가 약간 다를 수 있다.

2.7 Q&A

Q 10. 제가 만든 실수 계산 루틴은 매우 이상한 결과를 출력하고, 또 컴퓨터가 다르면 다른 결과가 나온다.

A. 문제가 간단하지 않겠지만, 디지털 컴퓨터에서 실수 처리는 정확하지 않을 수도 있다는 것을 알아두셔야 한다. 언더플로우(underow)가 일어날 수도 있으며, 오차가 점점 누적될 수도 있다. 따라서 실수 연산이 정확하지 않다는 것을 기억하기 바라며, 같은 이유로 두 실수가 같은지 비교하는 것은 좋지 않다. 이런 문제는 꼭 C 언어에만 국한된 것이 아니고, 모든 프로그래밍 언어에서 발생할 수 있다. 실수에 대한 어떤 부분들은 대개 "프로세서가 어떻게 처리하느냐"에 따라 다르다. 그렇지 않는 경우라면 적절한 기능을 수행할 수 있게 하기 위해 댓가가 큰 에뮬레이션을 사용한다.

2.7 Q&A

Q 11. Q 왜 C언어에는 지수를 (exponentiation) 계산하는 연산자가 없을까?

A. 대부분 프로세서에는 지수를 계산하는 명령(instruction)이 없기 때문이다. 대신 C 언어는 pow()라는 함수를 제공한다. 이 함수는 <math.h>에 선언되어 있습니다. 크기가 작고 0보다 큰 정수를 곱하는 것이 이 함수를 쓰는 것보다 더 효과적일 수도 있다.

2.8 실습 및 과제

- 실습 및 과제 진행
  - DCU Code : http://code.cu.ac.kr
  - 2 주차 실습
    - 코딩 2-1, 코딩 2-2, 코딩 2-3, 코딩 2-4, 코딩 2-5 (교재 p78-95)
  - 2 주차 과제
    - 2장 프로그래밍 연습 1 ~ 6번 (교재 p102)

2.9 참고문헌

- 자료형 : https://ko.wikipedia.org/wiki/자료형
- 정수형 : https://ko.wikipedia.org/wiki/정수형
- 실수형 변수 : https://ko.wikipedia.org/wiki/C_언어_실수형_변수
- char : https://ko.wikipedia.org/wiki/Char
- ASCII : https://ko.wikipedia.org/wiki/ASCII
- C 언어 변수 : https://ko.wikipedia.org/wiki/C_언어_변수

E N D