4장 기본 자료형

김명호

내용

- 자료형
- 선언문
- 수식과 문장
- 정수
- 문자
- 논리 값
- 실수
- Sizeof
- 복소수
- 일반적 산술 변환과 캐스트

자료형

- C 프로그램의 모든 변수는 자료형이 명시되어야 함
- 자료형(Data type) 또는 형(Type)
  - 기본 자료형
  - 사용자 자료형
- 메모리에는 모든 값들이 0또는 1의 비트열로 저장되지만 자료형에 따라 표현 방법이 다름

예제 프로그램

프로그램 4.1

```
#include <stdio.h>
int main(void){
    int i;
    i = 65;
    printf("정수 ? : %d\n", i);
    printf("실수 ? : %f\n", i);
    printf("문자 ? : %c\n", i);
    return 0;
}
```

프로그램 결과

```
정수 ? : 65
실수 ? : 0.000000
문자 ? : A
```

기본 자료형

_Bool			
char	signed char	unsigned char	
signed short int	signed int	signed long int	signed long long int
unsigned short int	unsigned int	unsigned long int	unsigned long long int
float	double	long double	
float _Complex	double _Complex	long double _Complex	
float _Imaginary	double _Imaginary	long double _Imaginary	

* 빨간색은 C99에서 추가된 형

기본 자료형(축약형)

_Bool			
char	signed char	unsigned char	
short	int	long	long long
unsigned short	unsigned	unsigned long	unsigned long long
float	double	long double	
float _Complex	_Complex	long double _Complex	
float _Imaginary	_Imaginary	long double _Imaginary	

기본 자료형(기능분류)

정수형	_Bool char short unsigned short	signed char int unsigned	unsigned char long unsigned long	long long unsigned long long
부동형	float float _Complex float _Imaginary	double _Complex _Imaginary	long double long double _Complex long double _Imaginary	
산술형	***정수형 + 부동형***			

선언문

- **모든 변수는 사용되기 전에 반드시 선언되어야 함**
- **선언문**

  자료형 식별자;

  자료형 식별자_목록; // 각 식별자는 콤마(,)로 분리됨
- **선언문 예**

```
int i;
float x, y;
```

선언문

- 배정 연산자를 사용하여 값을 저장할 수 있음

```
i = 2 * 5;     // 실행문
x = 5.0;
```

i	x	y
10	5.0	

```
y = i * x;
```

i	x	y
10	5.0	50.0

예제 프로그램

프로그램 4.2

```
#include <stdio.h>
int main(void){
   int num1, num2, num3, multiply;
   float fnum1, fnum2, fmultiply;

   printf("두 정수를 입력하세요 : ");
   scanf("%d%d", &num1, &num2);
   multiply = num1 * num2;
   fnum1 = num1;
   fnum2 = num2;
   fmultiply = fnum1 * fnum2;
   printf("%d * %d = %d \n", num1, num2, multiply);
   printf("%.2f * %.2f = %.2f\n", fnum1, fnum2, fmultiply);
   return 0;
}
```

프로그램 결과

```
두 정수를 입력하세요 : 4 3
4 * 3 = 12
4.00 * 3.00 = 12.00
```

선언문

- 선언한 변수를 사용하지 않아도 됨
- C90에서는 실행문 앞에 선언문이 있어야 함
- C99에서는 실행문과 선언문이 혼합될 수 있음
  - 변수가 사용되기 전에만 선언되면 됨

예제 프로그램

프로그램 4.3

```
#include <stdio.h>
int main(void){
    int num1, num2;
    printf("두 정수를 입력하세요 : ");
    scanf("%d%d", &num1, &num2);
    int multiply;
    multiply = num1 * num2;
    printf("%d * %d = %d \n", num1, num2, multiply);
    float fnum1, fnum2;
    fnum1 = num1;
    fnum2 = num2;
    float fmultiply;
    fmultiply = fnum1 * fnum2;
    printf("%.2f * %.2f = %.2f\n", fnum1, fnum2, fmultiply);
    return 0;
}
```

변수 선언의 목적

- 변수의 메모리 공간 확보

```
int num1, num2;                    // 각각 4 바이트 할당
```

- 올바른 연산자 선택

```
multiply = num1 * num2;            // 정수 *
fmultiply = fnum1 * fnum2;         // 실수 *
```

- 컴파일러에게 정보 전

```
fnum1 = num1;    // 정수 값을 실수로 변환하여 저장
```

수식과 문장

- **수식**
  - 상수, 변수, 함수 호출 그 자체
  - 수식과 연산자의 의미 있는 결합
  - 대부분의 수식은 값과 형을 가짐
- **예**
  - `30`
  - `'c'`
  - `a`
  - `a + b`
  - `c = a + b`
  - `5.0 * x - tan(9.0 / x)`

수식과 문장

- **문장**
  - 수식 뒤에 세미콜론이 오면, 수식은 문장이 됨
- **예**
  - `3.777;`
  - `a + b;`
  - `c = a + b;`

정수

- 정수
  - 부호있는 2의 보수로 표현
  - 첫 번째 비트 : 부호 비트 (1 : 음수, 0 : 양수)
  - 나머지 비트 : 2의 보수
  - 2의 보수
    - 양수 : 2진수
    - 음수 : 절대 값의 2진수의 1의 보수 + 1
    - 1의 보수 : 각 비트를 토글하는 것
      - (1->0, 0->1)

정수

- **정수의 표현**

0	1	2	.	.	.	31
부호	수의 2의 보수					

정수

- **341**
  - 부호 : 0
  - 341의 2진수

    0000000 00000000 00000001 01010101
  - 비트열

    0 0000000 00000000 00000001 01010101

    00000000 00000000 00000001 01010101

정수

- -341
  - 부호 : 1
  - -341의 절대값 2진수

    0000000 00000000 00000001 01010101
  - 1의 보수

    1111111 11111111 11111110 10101010
  - 1의 보수 + 1

    1111111 11111111 11111110 10101011
  - 비트열

    1 1111111 11111111 11111110 10101011

    11111111 11111111 11111110 10101011

연습 (8비트)

- 69
- -69

연습 (8비트)

- 69
  - 0100 0101
- -69

연습 (8비트)

- 69
  - 0100 0101
- -69
  - 1011 1011

int 부류 자료형

- int는 정수 기본형
- int 부류 자료형
  - 크기에 따른 분류
    - 2 바이트 : short, int(구형)
    - 4 바이트 : int, long
    - 8 바이트 : long long
    - * 모든 정수를 표현할 수 없음
  - 부호에 따른 분류
    - 음수 표현 가능 : signed
    - 음수 표현 불가능 : unsigned

int 부류 자료형

- short 자료형은 기억장소를 절약하고자 하는 경우에 사용
- long과 long long 형은 큰 정수 값을 다룰 때 사용
- unsigned 형의 변수는 음수가 아닌 정수를 표현할 때 사용함
- <limits.h> 헤더 파일에 크기 정의

int 자료형

- 값의 범위
  - 4 바이트 워드 컴퓨터
    - 최소 = -2^{31} = -2,147,483,648
    - 최대 = $+2^{31} - 1$ = +2,147,483,647
  - 2 바이트 워드 컴퓨터(구형 컴퓨터)
    - 최소 = -2^{15} = -32,768
    - 최대 = $+2^{15} - 1$ = +32,767

예제 프로그램

프로그램 4.4

```
#include <stdio.h>
#include <limits.h>
int main(void)
{
    printf("int 최소 값 : %d\n", INT_MIN);
    printf("int 최대 값 : %d\n", INT_MAX);
    return 0;
}
```

프로그램 결과

```
// 4 바이트 컴퓨터
int 최소 값 : -2147483648
int 최대 값 : 2147483647

// 2바이트 컴퓨터
int 최소 값 : -32768
int 최대 값 : 32767
```

int 부류 자료형

- 정수를 다룰 때 int가 기본형이지만, 필요에 따라 다양하게 선택할 수 있음

자료형	크기 (바이트)	범위	변환명세
short	2	$-2^{15} \sim 2^{15}-1$ (약 ±3만)	%hd, %hi, %hx, %ho
int	4	$-2^{31} \sim 2^{31}-1$ (약 ±21억)	%d, %i, %x, %o
long	4	$-2^{31} \sim 2^{31}-1$ (약 ±21억)	%ld, %li, %lx, %lo
long long	8	$-2^{63} \sim 2^{63}-1$ (약 ±922경)	%lld, %lli, %llx, %llo

int 부류 자료형

- **unsigned 형**

자료형	크기 (바이트)	범위	변환명세
unsigned short	2	0 ~ 2^{16}-1 (약 6만)	%hu, %hx, %ho
unsigned int	4	0 ~ 2^{32}-1 (약 42억)	%u, %x, %o
unsigned long	4	0 ~ 2^{32}-1 (약 42억)	%lu, %lx, %lo
unsigned long long	8	0 ~ 2^{64}-1 (약 1844경)	%llu, %llx, %llo

int 부류 자료형

- 정수 상수에 그 형을 명시하기 위해서는 접미사를 붙일 수 있음
- 접미사가 붙지 않은 정수 상수의 형은 int, long long, unsigned long long 중 하나 임
  - 시스템은 세 가지 형 중 그 정수 상수를 표현할 수 있는 첫 번째 것을 선택하여 그 정수 상수의 형으로 함
  - 예를 들어, 4 바이트 워드 컴퓨터에서 상수 2147483647은 int 형이고, 2147483648은 long long 형임

정수형 접미사

접미사	자료형	예제
u 또는 U	unsigned unsigned long unsigned long long	1024U 0xffffU
l 또는 L	long long long	1024L 0xffffL
ul 또는 UL	unsigned long unsigned long long	1024UL 0xffffUL
ll 또는 LL	long long	1024LL 0xffffLL
ull 또는 ULL	unsigned long long	1024ULL 0xffffULL

int 부류 자료형

프로그램 4.5

```
#include <stdio.h>
int main(void)
{
    long long v1, v2, v3;
    v1 = v2 = 9000000000000000000LL;        // 900 경
    v3 = v1 + v2;          // 900 경 + 900 경 = 1800경?
    printf("%lld + %lld = %lld\n", v1, v2, v3);
    return 0;
}
```

프로그램 결과

```
9000000000000000000 + 9000000000000000000 = -4467440737095516 16
```

int 부류 자료형

- **정수 오버플로**
  - 값의 범위를 초과할 때 발생 (주의 필요)
  - 정수 오버플로가 발생해도 프로그램은 계속 수행되지만, 논리적으로 부정확한 값이 계산됨

문자

- 문자도 컴퓨터에 저장될 때에는 0과 1의 비트열로 저장됨
- 문자를 비트열로의 변환은 표준이 있음
  - ASCII
  - EBCDIC

ASCII 코드

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	np	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

ASCII 코드

- ASCII 코드에서 문자 상수와 대응되는 정수 값

문자 상수	'a'	'b'	'c'	...	'z'
대응하는 값	97	98	99	...	122
문자 상수	'A'	'B'	'C'	...	'Z'
대응하는 값	65	66	67	...	90
문자 상수	'0'	'1'	'2'	...	'9'
대응하는 값	48	49	50	...	57
문자 상수	'&'	'*'	'+'		
대응하는 값	38	42	43		

ASCII 코드

- 주의
  - 숫자 문자와 그에 대응되는 ASCII 값은 완전히 다름
    - '2'의 값은 2가 아니라 50
  - 하지만 연속적으로 ASCII 값이 배정됨
    - '0' : 48, '1' : 49, '2' : 50, . . .
  - 알파벳도 연속적인 값을 가짐
    - 'a' : 97, 'b' : 98, 'c' : 99, . . .
    - 문자와 단어를 사전적 순서로 정렬할 때 유용

문자

- 'a', '+'와 같은 문자 상수는 char 형이 아니라 int 형임
- 모든 정수적 형의 변수는 문자를 표현하는 데 사용될 수 있음

char 자료형

- char 형 변수는 문자와 정수 값을 저장하는 데 사용됨
- char는 메모리의 1 바이트에 저장됨
  - 256개의 값을 저장할 수 있음
  - 이 값들 중 일부만이 실제 인쇄 문자임
    (소문자, 대문자, 숫자, 마침표, 특수문자 등)
  - 공백, 탭, 개행 문자 같은 여백도 있음

char 자료형

- char, signed char, unsigned char
- char는 시스템에 따라 signed char 또는 unsigned char와 같음
- 세 가지의 char 형은 각각 1 바이트에 저장됨 (256개의 값 표현)
- signed char 형의 값의 범위 : -128 ~ 127
- unsigned char 형의 범위 : 0 ~ 255
- char 형은 정수 값을 저장하기 위해 사용될 수도 있음

예제 프로그램

프로그램 4.6

```
#include <stdio.h>
int main(void)
{
    char c = 'a';
    printf("c = %c\n", c);
    printf("c = %d\n", c);
    return 0;
}
```

프로그램 결과

```
c = a
c = 97
```

예제 프로그램

프로그램 4.6

```
#include <stdio.h>
int main(void)
{
    int c = 'a';
    printf("c = %c\n", c);
    printf("c = %d\n", c);
    return 0;
}
```

프로그램 결과

```
c = a
c = 97
```

char 자료형

- 와이드 문자(Wide character)
  - 문자 집합의 크기가 256보다 큰 언어를 다룰 때 사용
  - wchar_t 형 : int 부류 자료형 중 하나로 표현
  - 일반적인 char 형을 네로우 문자라고도 함
- 다중 바이트 문자
  - char 형을 여러 개 사용하여 한 문자 표현하는 방법
  - 와이드 문자의 크기는 일정한 반면, 다중 바이트 문자의 크기는 표현 문자에 따라 다름

논리 값과 _Bool 형

- 논리 값을 다루기 위해 _Bool 형이 C99에 추가됨
- 참과 거짓을 표현하기 위해 사용
  - 거짓 : 0
  - 참 : 1
- <stdbool.h>에 유용한 매크로 정의
  - true : 1
  - false : 0
  - bool : _Bool

예제 프로그램

프로그램 4.8

```
#include <stdio.h>
#include <stdbool.h>
int main(void){
    bool a, b, c;        // _Bool  a, b, c;
    a = true;            // a = 1;
    b = false;           // b = 0;
    c = 324;
    printf("a = %d, b = %d, c = %d\n", a, b, c);
    return 0;
}
```

프로그램 결과

```
a = 1, b = 0, c = 1
```

실수 표현

- 부동소수로 표현
- 모든 수를 지수 형태로 변환하여 부호, 지수, 가수부로 나누어 저장
  - $4.0 \rightarrow 1.0 \times 2^2$
    - 부호 : +
    - 지수 : 2
    - 가수 : 1.0
- IEEE 부동소수 : IEC 60559 표준
  - 단정밀도(32비트), 배정밀도(64비트), 단확장 정밀도(43비트 이상), 배확장 정밀도(70비트 이상)

실수 표현

- **단정밀도**

0	1 ~ 8	9 ~ 31
부호	지수부	가수부

- **배정밀도**

0	1 ~ 11	12 ~ 63
부호	지수부	가수부

실수

- 정수와 실수 연산이 다른 이유
  - 컴퓨터에서 정수와 실수의 표현방법이 다르기 때문
  - 정수 : (부호있는) 2의 보수
  - 실수 : IEEE 부동소수 표준

  - $3 + 4$
  - $3 \times 10^5 + 4 \times 10^{-2}$

메모리 내용

```
. . .
1111 1011 1001 1010 1100 1010 0110 0001
0011 1011 1001 1010 1001 1010 0100 0111
1011 1111 1001 1110 1111 1000 1100 1101
0011 1011 1001 1010 1100 1010 0100 0001
0101 1000 1111 0000 1100 1010 0101 0000
0011 1000 0000 1010 1110 0011 0100 1111
. . .
```

0011 1011 1001 1010 1100 1010 0100 0001

- **int**

  0011 1011 1001 1010 1100 1010 0100 0001

- **flot**

  0011 1011 1001 1010 1100 1010 0100 0001

- **char**

  0011 1011 1001 1010 1100 1010 0100 0001

0011 1011 1001 1010 1100 1010 0100 0001

- **int**

  0 011 1011 1001 1010 1100 1010 0100 0001

- **flot**

  0 011 1011 1001 1010 1100 1010 0100 0001

- **char**

  0 011 1011 1001 1010 1100 1010 0100 0001

0011 1011 1001 1010 1100 1010 0100 0001

- **int**

  0 011 1011 1001 1010 1100 1010 0100 0001

  => 1000000069

- **flot**

  0 011 1011 1001 1010 1100 1010 0100 0001

  => 0.004724

- **char**

  0 011 1011 1001 1010 1100 1010 0100 0001

  => (65) A

부동형

- 실수 값을 다루기 위해 사용함
- float, double, long double
  - float : 단정밀도
  - double : 배정밀도
  - long double : 시스템 종속적
- 부동형으로 모든 실수를 표현할 수 없음

부동형

- **부동형에 배정될 수 있는 값은 *정밀도*와 *범위*라는 속성으로 기술됨**
  - 정밀도 : 부동형이 표현할 수 있는 유효숫자 수
  - 범위 : 부동형이 저장할 수 있는 가장 큰 양수와 가장 작은 양수

부동형

- float 형
  - 정밀도 : 대략 유효숫자 6자리
  - 범위 : 대략 10^{-38}에서 10^{+38}
- double 형
  - 정밀도 : 대략 유효숫자 15자리
  - 범위 : 대략 10^{-308}에서 10^{+308}

부동형

자료형	크기 (바이트)	범위	변환명세
float	4	10^{-38} ~ 10^{+38}	%f, %e, %g, %a
double	8	10^{-308} ~ 10^{+308}	%f, %e, %g, %a
long double	?	?	%Lf, %Le, %Lg

예제 프로그램

프로그램 4.9

```
#include <stdio.h>
int main(void)
{
    float x = 8.88888888888888888888;    // 20개의 8
    printf("x = %.20f\n", x);
    return 0;
}
```

프로그램 결과

```
x = 8.88888931274414062500
```

예제

- **x = 123.45123451234512345;**
  - x의 실제 값
    - double : $0.12345123451234 5 \times 10^3$
    - float : 0.123451×10^3
- **주의**
  - 모든 실수를 다 표현할 수는 없다.
  - 정수 산술 연산과 달리 부동형 산술 연산은 정확하지 않다.

부동형 상수 표기법

- 십진 표기법
  - 1.0 또는 1. 또는 .0001
- 지수 표기법
  - 10진 부동형 상수
    - 1.234567e5 (= 1.234567 X 10^5)
  - 16진 부동형 상수
    - 0x1.234567p5 (= 0x1.234567 X 2^5)

부동형 상수 표기법

- **올바른 부동형 상수**
  - 3.14159, 314.159e-2F, 0e0, 1.
- **잘못된 부동형 상수**
  - 3.14,159
  - 314159
  - .e0
  - -3.14159

부동형 접미사

접미사	자료형	예제
f, F	float	10.24F 1.024e3f 0x1p10f
없음	double	10.24 1.024e3 0x1p10
l, L	long double	10.24L 1.024e3L 0x1p10L

sizeof

- 피연산자로 명시된 객체를 저장하는 데 필요한 바이트 수를 알아내기 위해 사용
  - 피연산자로는 자료형이나 수식이 올 수 있음
- 사용법
  - sizeof(int) : int의 크기 알려줌; 괄호를 생략할 수 없음
  - sizeof(c) : c의 크기를 알려줌; 괄호 생략 가능
  - sizeof(a + b) : a + b의 형의 크기를 알려줌
  - sizeof a + b : (sizeof a) + b

예제 프로그램

프로그램 4.10

```
#include <stdio.h>
int main(void){
   printf("기본 자료형의 크기\n\n");
   printf(" char:%3d 바이트 \n", sizeof(char));
   printf(" short:%3d 바이트\n", sizeof(short));
   printf(" int:%3d 바이트\n", sizeof(int));
   printf(" long:%3d 바이트\n", sizeof(long));
   printf(" long long:%3d 바이트\n", sizeof(long long));
   printf(" unsigned:%3d 바이트\n", sizeof(unsigned));
   printf(" float:%3d 바이트\n", sizeof(float));
   printf(" double:%3d 바이트\n", sizeof(double));
   printf("long double:%3d 바이트\n", sizeof(long double));
   return 0;
}
```

프로그램 결과

- 컴퓨터 마다 다를 수 있지만 다음 조건은 만족함

```
sizeof(char)    = 1
sizeof(short)  <=  sizeof(int)      <=  sizeof(long)
sizeof(signed)  =  sizeof(unsigned)  =  sizeof(int)
sizeof(float)   <  sizeof(double)    <  sizeof(long double)
```

복소수

- 실수부와 허수부로 이루어짐

  2.5 + 4.2i

- _Complex 형 사용
  - float _Complex
  - double _Complex
  - long double _Complex
- _Complex 형 변수는 두 개의 부동형 실수 값을 가짐
- 프로그램에서 허수부는 i(I)로 표현함

예제 프로그램

프로그램 4.11

```
#include <stdio.h>
int main(void){
    double _Complex a = 2.5 + 4.2i;
    double _Complex b = 2.8I + 3.2;

    printf("double _Complex 크기: %d\n\n",
            sizeof(double _Complex));
```

예제 프로그램

프로그램 4.11

```
printf("복소수 a : %4.2f %4.2f\n", a);
printf("복소수 b : %4.2f %4.2f\n\n", b);

printf("복소수 a : %4.2f%+4.2fi\n", a);
printf("복소수 b : %4.2f%+4.2fi\n\n", b);
```

예제 프로그램

프로그램 4.11

```
    printf("a + b = %4.2f%+4.2fi\n", a + b);
    printf("a - b = %4.2f%+4.2fi\n", a - b);
    printf("a * b = %4.2f%+4.2fi\n", a * b);
    printf("a / b = %4.2f%+4.2fi\n", a / b);
    return 0;
}
```

프로그램 결과

```
double _Complex 크기: 16

복소수 a : 2.50 4.20
복소수 b : 3.20 2.80

복소수 a : 2.50+4.20i
복소수 b : 3.20+2.80i

a + b = 5.70+7.00i
a - b = -0.70+1.40i
a * b = -3.76+20.44i
a / b = 1.09+0.36i
```

복소수

- <complex.h>에는 complex 매크로와 다양한 복소수 함수가 정의되어 있음
  - #define complex _Complex
  - creal() : 복소수의 실수부
  - cimag() : 복소수의 허수부

예제 프로그램

프로그램 4.12

```
#include <stdio.h>
#include <complex.h>
int main(void){
    double complex a = 2.5 + 4.2i;

    printf("a : %4.2f%+4.2fi\n", creal(a), cimag(a));

    return 0;
}
```

프로그램 결과

```
a : 2.50+4.20i
```

일반적 산술 변환

- 산술 수식은 값과 형을 가짐
- 산술 수식의 피연산자들의 형이 다르면 피연산자의 형을 변환하여 일치 시킴
- 일반적 산술 변환은 컴파일러에 의해 자동적으로 일어남
  - 컴파일러는 선언문을 참조하여 이러한 일을 수행함
  - 자동 형 변환(Automatic type conversion)
  - 묵시적 형 변환(Implicit type conversion)

일반적인 산술 변환

- **정수 승격**
  - 정수적 수식에서 일반적 산술 변환의 한 단계로 일어남
  - _Bool, char, signed char, unsigned char, signed short, unsigned short, 비트 필드 형은 산술 수식에서 int나 unsigned int로 변환
  - 예

```
char a, b, c;
c = a + b;      // a와 b는 int로 정수 승격됨
```

일반적인 산술 변환

- 기본 규칙 : 큰 쪽으로 형 변환
- 규칙

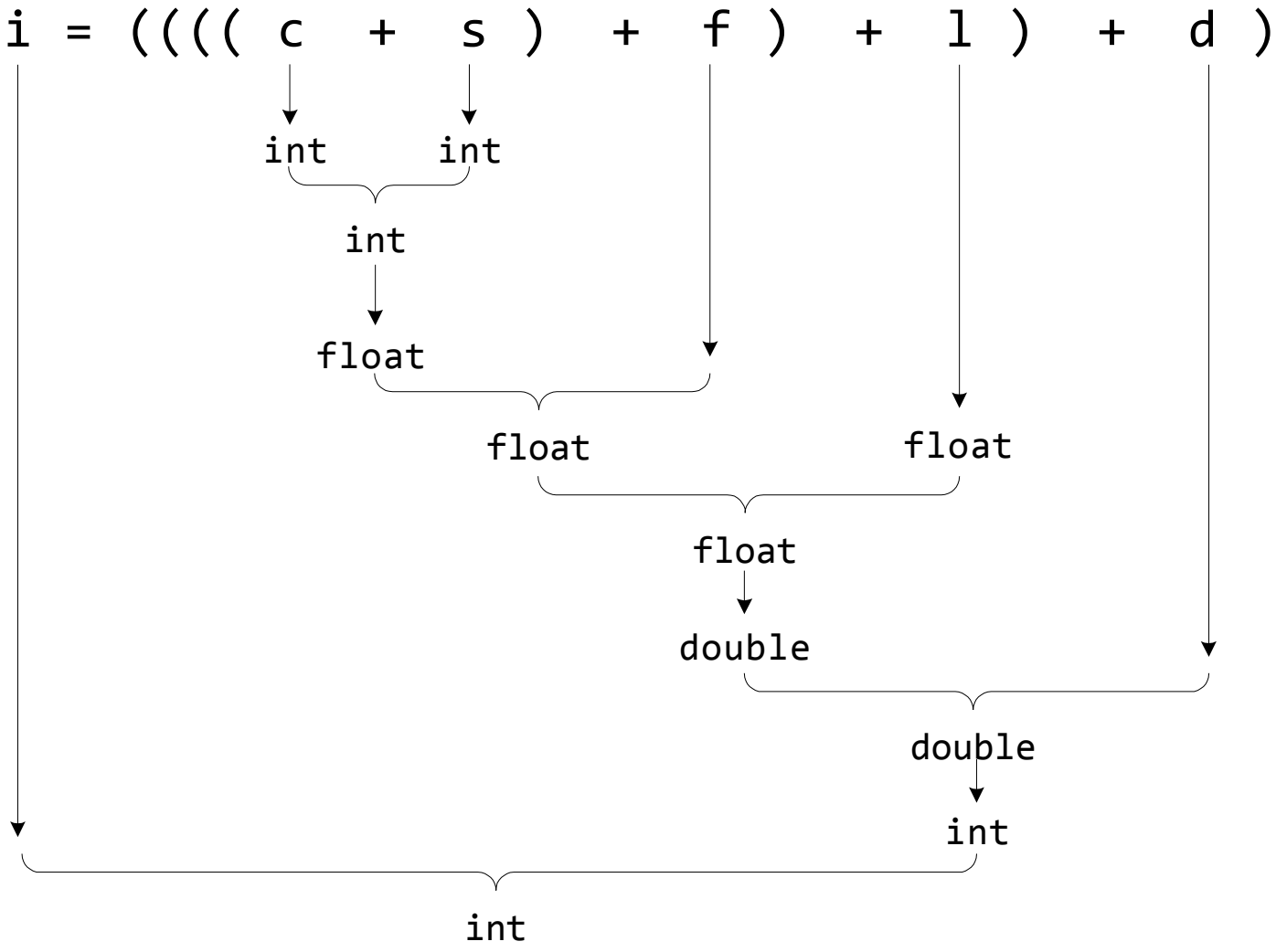
```
if (한 피연산자가 long double) 다른 피연산자를 long double로 변환
else if (한 피연산자가 double) 다른 피연산자를 double로 변환
else if (한 피연산자가 float) 다른 피연산자를 float으로 변환
else {
   두 피연산자에 정수 승격 적용
   if (두 피연산자의 형이 같음) 자동 변환 종료
   else if (한 피연산자가 unsigned long long)
            다른 피연산자를 unsigned long long으로 변환
   else if (한 피연산자가 long long) 다른 피연산자를 long long으로 변환
   else if (한 피연산자가 unsigned long) 다른 피연산자를 unsigned long으로 변환
   else if (한 피연산자가 long) 다른 피연산자를 long으로 변환
   else if (한 피연산자가 unsigned) 다른 피연산자를 unsigned로 변환
}
```

일반적인 산술 변환 예제

```
char c;
unsigned short s;
int i;
long l;
float f;
double d;

i = c + s + f + l + d;
```

일반적인 산술 변환 예제


i = ((((c + s) + f) + l) + d)
int
int
int
float
float
float
float
double
double
int
int

자동 변환 예제

선언			

`char c;` `short s;` `int i;`
`long l;` `unsigned u;` `unsigned long ul;`
`float f;` `double d;` `long double ld;`			

수식	**형**	**수식**	**형**
`c - s / i`	`int`	`u * 7 - i`	`unsigned`
`u * 2.0 - i`	`double`	`f * 7 - i`	`float`
`c + 3`	`int`	`7 * s * ul`	`unsigned long`
`c + 5.0`	`double`	`ld + c`	`long double`
`d + s`	`double`	`u - ul`	`unsigned long`
`2 * i / l`	`long`	`u – l`	*system-dependent*

캐스트

- 명시적인 변환이 필요할 때가 있음

```
int sum = 9, num = 10;
float avg;
avg = sum / num;        // / : 정수 나누기
```

- 캐스트 연산자
  - 괄호 안의 자료형

    (int), (float)
  - 형을 변환하고자하는 수식 앞에 붙임

    (float) sum

캐스트

- **캐스트 예제**

```
avg = (float) sum / (float) num;
avg = (float) sum / num;
(double) (num = 10)
```

- **잘못된 캐스트**

```
(double) num = 10;
```