컴퓨터 프로그래밍 I
(CSE2003-3)

Mon/Wed 16:30-17:45 pm
Week 1

Expression (표현식)
Data type conversion (자료형 변환)

Calculate a circle's area and circumference

```
#include<stdio.h>


int main(void)
{
    float circ, area, radius;

    printf("\nPlease enter the value of the radius: ");
    scanf("%f", &radius);

    circ = 2  * 3.14  * radius;          // 원주 계산식
    area = 3.14 * radius * radius;       // 넓이 계산식

    printf("\nRadius is :              %10.2f", radius);
    printf("\nCircumference is : %10.2f", circ);
    printf("\nArea is :                %10.2f\n", area);

    return 0;
}
```

- 이 프로그램은 입력 받은 값을 반지름으로 하는 원의 원주와 넓이를 구하는 프로그램이다.
- 각각의 값은 전체 폭 10자리, 그 중 소수점 아래 2자리로 맞추어 출력된다.

계산식 (Expression/표현식)
계산식은 표현식의 일종

<실행결과>

```
Please enter the value of the radius: 3

Radius is :              3.00
Circumference is :      18.85
Area is :               28.27
```

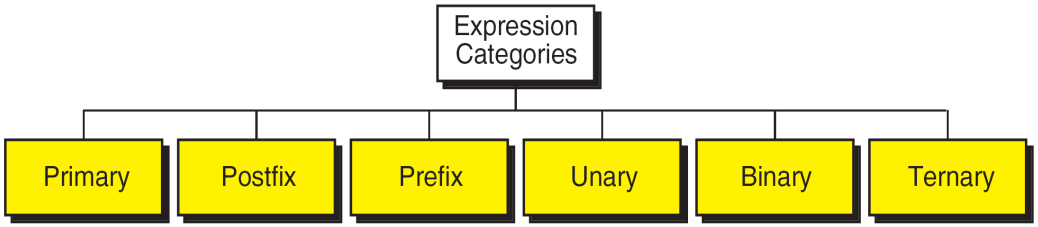
Expression (표현식)

- 프로그래밍 언어에서 피연산자(operand)들과 연산자(operator)들로 구성된 식.
  - Operator(연산자) : =, +, -, *, / 등.
  - Operand (피연산자): operator를 가지고 계산되는 상수 또는 변수.
- Expression(표현식)은 항상 결과(반환) 값을 갖는다.
  - 표현식의 예: 2 + 5 다항 연산자 결과값: 7

    -a 단항 연산자. 결과값: -a

    2 + 5 * 7 결과값: 37

    x = x + 1 결과값: x+1
- 표현식의 구분



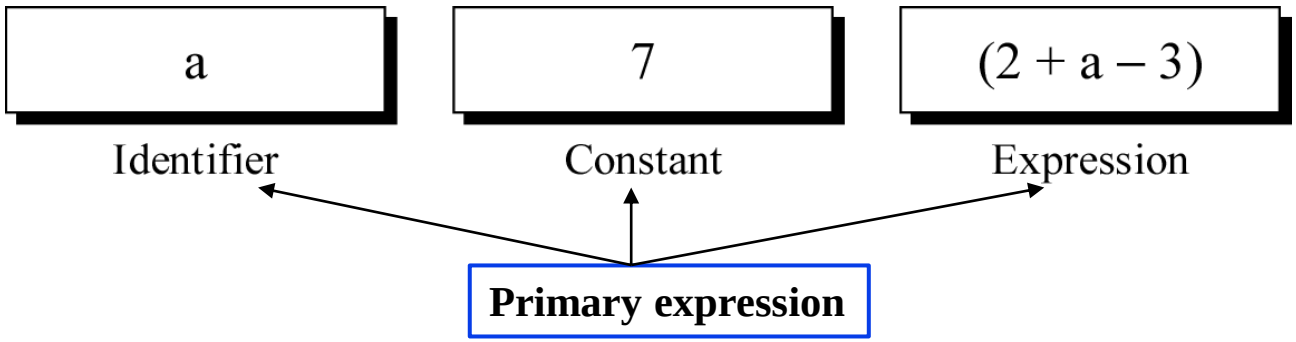
연산자	연산의 예	의미
=	a=20	대입
+	4+3	덧셈
-	4-3	뺄셈
*	4*3	곱셈
/	4/3	나눗셈
%	4%3	나머지

Expression (표현식)

- **Primary expression**

  : operator가 없이 오직 하나의 operand로 이루어진 가장 기본적인 형태

  - 식별자 (Identifier)
    - 식별자 예 : a, b12, price, calc, INT_MAX 등
  - 상수 (Constants)
    - Constants 예 : 5(정수), 123.45(실수), 'A'(문자) 등
  - 괄호식 (Parenthetical Expression)
    - 괄호식 예 : (2*3), (23+b*6) 등



Expression (표현식)

<원의 넓이와 원주를 구하는 프로그램>

```
#include <stdio.h>

int main (void)
{
    float circ;
    float area;
    float radius;

    printf("\nPlease enter the value of the radius:");
    scanf("%f", &radius);

    circ = 2 * 3.14 * radius;
    area = 3.14 * (radius * radius);

    printf("\nRadius is :    %10.2f", radius);
    printf("\nCircumference is :%10.2f", circ);
    printf("\nArea is :      %10.2f", area);

    return 0;
}
```

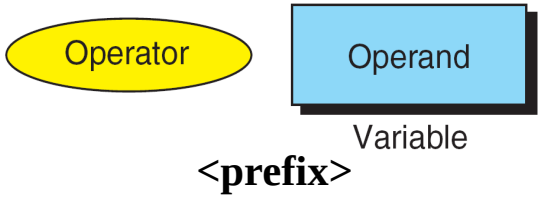
식별자(identifier)

상수(constant)

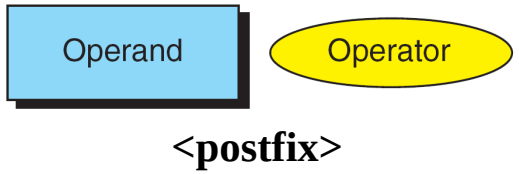
괄호식

특별한 연산자(operator): ++, -- (증감연산자)

- **Prefix expression**
  - 형식 : operator가 operand 앞에 나옴
  - 예 : -a, ++a, --a



- **Postfix expression**
  - 형식 : operator가 operand 뒤에 나옴
  - 예 : a++, a--



- ++a와 a++의 공통점
  - ‘a = a + 1’ 또는 ‘a += 1’ 의 결과와 같은 효과를 낸다.

특별한 연산자(operator): ++, --

```
#include <stdio.h>

int main (void)
{
    int a;

    a = 4;
    ++a;
    printf("Value of ++a : %d\n", a);

    a = 4;
    a++;
    printf("Value of a++ : %d\n", a);

    a = 4;
    a = a + 1;
    printf("Value of a+1 : %d\n", a);

    return 0;
}
```

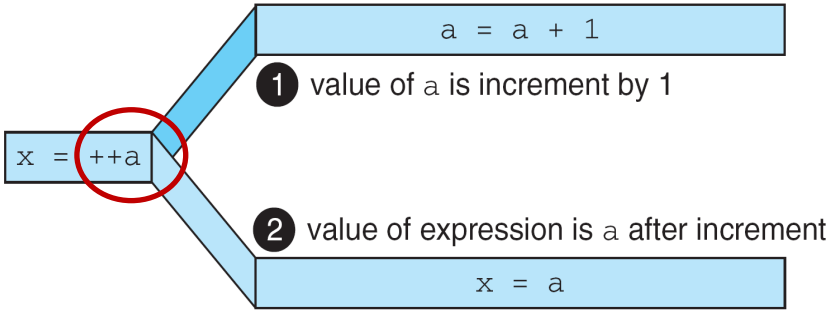
프로그램의 실행 결과는?

```
Value of ++a :  5

Value of a++ :  5

Value of a+1 :  5
```

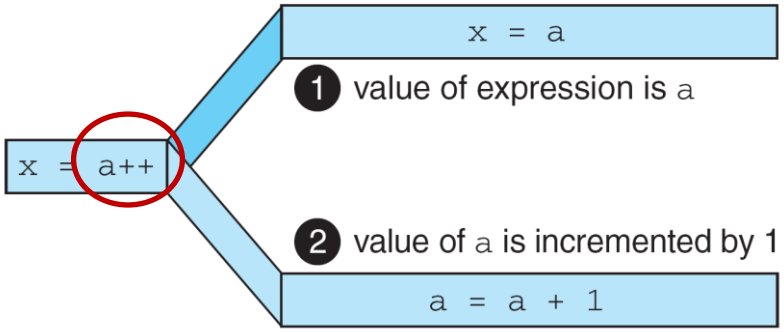
특별한 연산자(operator): ++, --

- ++a 과 a++의 차이
  - prefix와 postfix는 증가/감소 연산을 수행하는 시점이 다름
    - prefix의 경우, 변수 a의 값을 읽기 전에 먼저 ++ 연산을 수행
    - postfix의 경우, 변수 a값을 읽고 난 후에 ++ 연산을 수행

<prefix>



<postfix>



특별한 연산자(operator): ++, --

<prefix>

```
#include <stdio.h>

int main (void)
{
    int a, x;

    a = 4;
    printf("Value of a    :%d\n", a);
    printf("Value of ++a  :%d\n", ++a);
    printf("New value of a:%d\n", a);

    a = 4;
    x = ++a;
    printf("Value of x    :%d\n", x);

    return 0;
}
```

프로그램의 실행 결과는?

```
Value of a    :4
Value of ++a  :5
New value of a:5
Value of x    :5
```

<postfix>

```
#include <stdio.h>

int main (void)
{
    int a, x;

    a = 4;
    printf("Value of a    :%d\n", a);
    printf("Value of a++  :%d\n", a++);
    printf("New value of a:%d\n", a);

    a = 4;
    x = a++;
    printf("Value of x    :%d\n", x);

    return 0;
}
```

프로그램의 실행 결과는?

```
Value of a    :4
Value of a++  :4
New value of a:5
Value of x    :4
```

특별한 연산자(operator): ++, --

- prefix나 postfix 형식을 사용한 증감연산식과 infix 형태의 일반 산술연산식의 비교

```
#include <stdio.h>

int main (void)
{
    int x = 10;

    printf("%d\n", x++);
    printf("%d\n", x);
    printf("%d\n", --x);
    printf("%d\n", x);

    return 0;
}
```

printf	X
	10
10	11
11	11
10	10
10	10

```
#include <stdio.h>

int main (void)
{
    int x = 10;

    printf("%d\n", x+1);
    printf("%d\n", x);
    printf("%d\n", x-1);
    printf("%d\n", x);

    return 0;
}
```

printf	X
	10
11	10
10	10
9	10
10	10

특별한 연산자(operator): ++, --

- 증감연산식 사용 이유
  - 증감연산자를 이용하면 프로그램 형태가 간결
  - 기계어 코드와 일대일 대응되므로 실행속도가 빨라짐

- 주의점
  - 연산자의 위치에 따라 evaluation value가 다르므로 주의 요구
  - 증감연산자는 ++, -- 자체가 연산자 기호 ➔ 중간에 공백이 들어가면 안됨
  - 증감연산자는 피연산자로 변수만 사용 가능. 상수나 괄호식과 같은 일반 수식을 피연산자로 사용할 수 없음

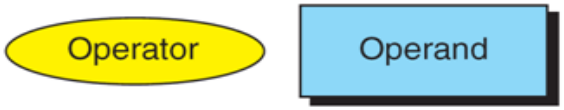
특별한 연산자(operator): ++, --

문제) 잘못된 표현식은?

```
#include <stdio.h>

int main (void)
{
    int a, b, c;

    a = 7;
    b = 77++;          /* 상수에는 증감 연산자를 사용할 수 없다 */
    c = ++a;
    c--;
    --(a * c - 1);     /* 일반 수식에는 증감 연산자를 사용할 수 없다 */
    (b + a)++;         /* 일반 수식에는 증감 연산자를 사용할 수 없다 */

    return 0;
}
```

Expression (표현식)

- **Unary expression**
  - 형식 : 연산자 – 피연산자 (operator - operand)
  - 예
    - +a, -a
    - sizeof(int), sizeof x, sizeof -12.5
    - (float)x 등.



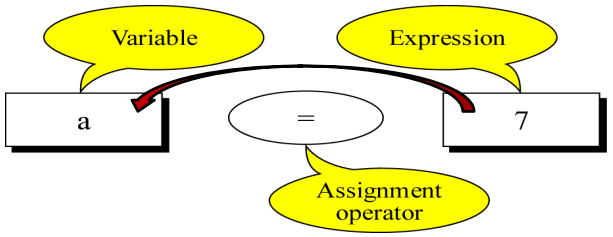
<Unary expression>

<Binary expression>

- **Binary expression**
  - 형식 : 피연산자 – 연산자 – 피연산자 (operand – operator - operand)
  - Operator가 operand사이에 있음
  - 예
    - a+7, 3+4, b-11 등 ← 덧셈 연산자(+, -)
    - 10*12, a /4, 5%2 등 ← 곱셈 연산자(*, /, %)
      - %(modulo)는 나머지를 구하는 연산자

Assignment expressions (대입문)

- 수학에서는 "우측의 값과 좌측의 값이 같다"라는 의미이나,
  C 언어에서는 "우측의 값을 좌측의 저장 장소에 저장하라"라는 의미.
- assignment operator '='의 좌측은 반드시 single variable이어야 한다.



x = x + y;
⇒ x += y;

- assignment 종류
  - Simple assignment
  - 예 : a=5, b=x+1, i=i+1등
  - Compound assignment
  - 예) x += y, x *= y, x /= y 등

Contents of Variable x	Contents of Variable y	Expression	Value of Expression	Result of Expression
10	5	x = y + 2	7	x = 7
10	5	x = x / y	2	x = 2
10	5	x = y % 4	1	x = 1

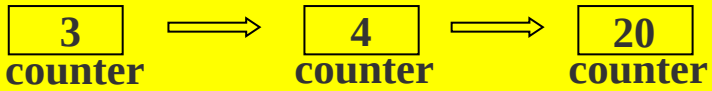
Compound Expression	Equivalent Simple Expression
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y
x += y	x = x + y
x -= y	x = x - y

Assignment expressions (대입문)

- **Destination of an assignment**
  - 대입 명령은 항상 대입 연산자(assignment operator)의 오른쪽에서 왼쪽으로.

<원의 넓이와 원주를 구하는 프로그램>

```
#include <stdio.h>

int main (void)
{
    float circ;
    float area;
    float radius;
    float temp1, temp2;

    radius = 9;                          /*변수(variable) radius에 값(value) 9를 대입*/
    9 = radius;                          /*9는 올바른 변수 이름이 아니므로 잘못된 표현*/

    circ = 2 * 3.14 * radius;            /*변수 원주를 구한 결과값을 변수 circ에 대입*/
    circ / 2 = 3.14 * radius;            /*circ/2는 변수가아닌 표현식이므로 잘못된 표현/

    counter = 3;
    counter = counter + 1;
    counter = 5 * counter;

    area = temp2;                        /*변수 temp2가 갖는 값을 변수 area에 대입*/
    temp2 = area;                        /*변수 area가 갖는 값을 변수 temp2에 대입*/

    temp1 = temp2 = 0;
    temp1 = (temp2 = 0);

    return 0;
}
```



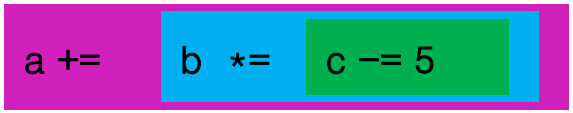
대입 명령은 항상 연산자의 오른쪽에서 왼쪽으로 대입되므로 옆의 두 명령은 같은 의미를 갖는다.
예 : A = B = C = 0; or A = (B = (C = 0);

Precedence & Associativity

- **Precedence : 다른 연산자들 사이의 우선 순위**
  - 수학에서도 *(곱하기) 와 /(나누기) 연산자가 +(더하기) 나 – (빼기) 연산자보다도 우선 순위가 더 높은 것처럼 C언어에서도 연산자 사이에 우선순위가 있다.
  - Precedence의 예 :
    - 2 + 3 * 4 ➔ (2 + (3 * 4))
    - - b ++ ➔ (- (b++))

- **Associativity : 같은 순위의 연산자들 사이의 우선 순위**
  - Associativity의 예 :
    - Left-to-Right (Simple Expression)
      - 3 * 8 / 4 % 4 * 5 ➔ ((((3 * 8) / 4) % 4) * 5)

    - Right-to-Left (Compound Assignment)
      - a += b *=c -= 5 ➔ (a += (b *= (c -= 5)))
        ➔ (a = a + (b = b * (c = c – 5)))



Precedence & Associativity code convention

Precedence and Associativity of Operators in C

Precedence	Associativity	Operators			
1	L → R	()	[]	->	.
2	R → L	!	~	++	--
		(*type*)	- (unary)	+ (unary)	sizeof
		* (deference)	& (address of)		
3	L → R	* (multiply)	/	%	
4	L → R	+	-		
5	L → R	<<	>>		
6	L → R	<	<=	>	>=
7	L → R	==	!=		
8	L → R	& (bitwise and)			
9	L → R	^			
10	L → R	|			
11	L → R	&&			
12	L → R	||			
13	R → L	? :			
14	R → L	=	*=	/=	%=
		+=	-=	<<=	>>=
		&=	|=	^=	
15	L → R	,			

Precedence & Associativity

```
#include <stdio.h>

int main (void)
{
    int a = 10;
    int b = 20;
    int c = 30;

    printf("a * b + c is       :%d\n", a * b + c);
    printf("a * (b + c) is     :%d\n", a * (b + c));
    printf("3 * -8 + 34 % 5 is :%d\n", 3 * -8 + 34 % 5);

    return 0;
}
```

출력되는 결과는?

```
a * b + c is       :230
a * (b + c) is     :500
3 * -8 + 34 % 5 is :-20
```

Data Type Conversion (자료형 변환)

- **Type Conversion**
  - 일반적으로 C 언어의 연산식에서 여러 피연산자의 자료형이 서로 다른 경우, 하나의 통일된 자료형으로 자동 변환하여 연산을 수행
  - **Implicit type conversion (coercion)** :
    C 컴파일러가 판단하여 자동으로 데이터형을 변형하는 경우.
    - 다음과 같은 두가지 경우 C 컴파일러가 판단하여 자동 형 변환을 수행한다.
      1. 수식에서 데이터형이 혼합되어 사용되었을 때 값을 자동 변환.
      2. 특정한 데이터형의 변수에 다른 데이터형의 값을 대입할 때, 값을 자동 변환.

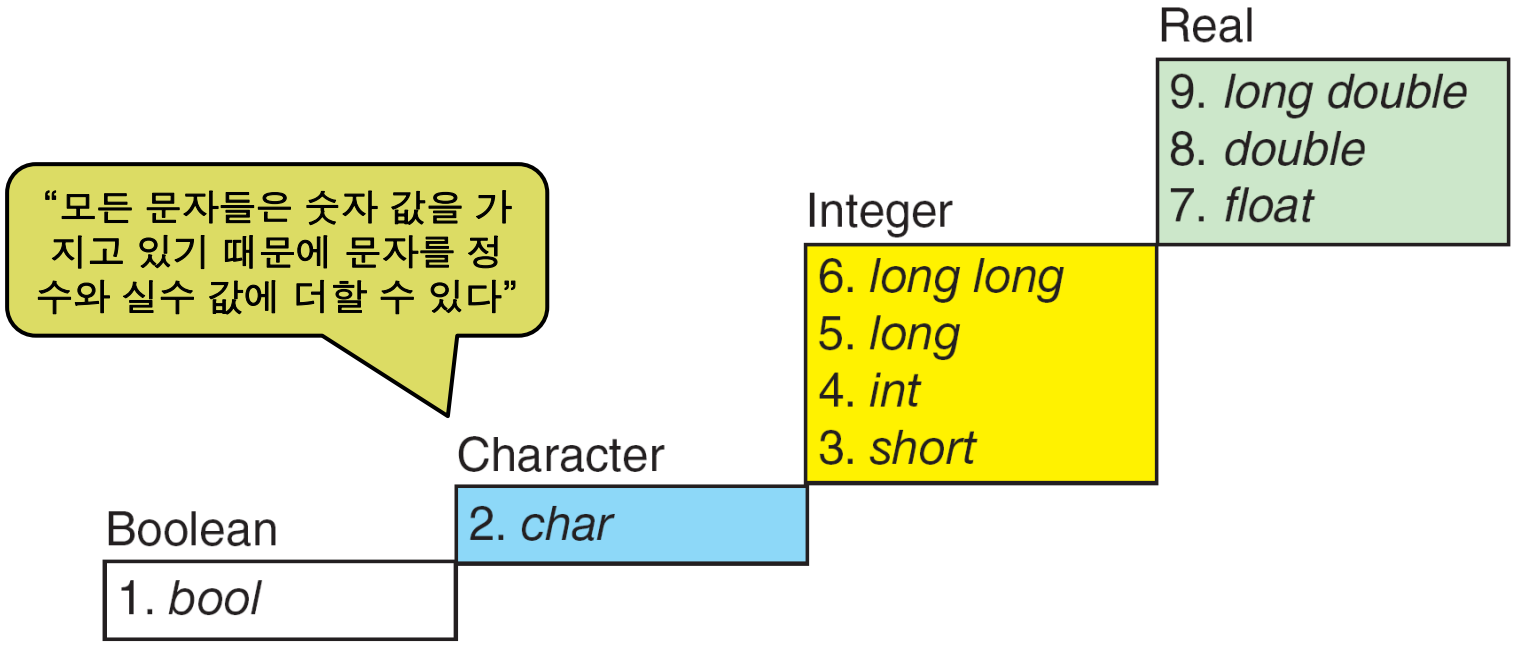
```
int x = 1;
char y = 'a';     // 'a'== 01100001(61_16) == 97_10
x= y;             // x = 97 (character y의 값 'a' 의 ASCII code 값을 정수로 변환한 형태)
```

  - **Explicit type conversion (cast)** : 프로그래머가 형 변환자(cast operator)를 사용하여 강제적으로 변형하는 경우.

```
float x = 1123.654;
int y ;
y = (int)x;       // y = 1123 (x의 값에 소수점 아래를 버리면서 int형으로 변환한 후 대입.)
```

Implicit type conversion (Coercion)

- 서로 다른 데이터 type의 operand들이 혼합되어 사용될 때
  - 수식 계산에서 형 변환은 보통 범위가 큰 데이터형으로 자동 변환
  - Conversion Rank
    - 각각의 데이터 type들에 대해 계급을 할당
    - Ex) *long double* 타입의 실수는 *long* 타입의 정수보다 높은 계급을 갖고 있다.



Implicit type conversion (Coercion)

- 형 확장(Promotion)
  - 하나 이상의 데이터 type이 혼합되어 식을 구성할 때
  - 가장 큰 데이터 type에 맞춰 자동으로 형 변환이 이루어지는 규칙을 적용

```
#include <stdio.h>

int main (void)
{
     char ch;
     int i;
     float f;
     double d;

     ch = 'A';
     i = 10;
     f = 1.5;
     d = ch * i * f + 300;

     printf("result = %lf\n", d);

     return 0;
}
```

1) ch * i ➔ ch는 int형으로 확장되어 i와 곱셈 수행된다. 여기서 ch는 대문자 A 인데 이는 ASCII 코드값 65 로 계산된다.

2) * f ➔ 1)에서 계산된 결과는 float형으로 확장되어 f와 계산 수행된다.

3) + 300 ➔ 300도 float형으로 확장되어 2)의 결과에 더해진다.

4) 계산의 결과는 double형으로 확장되어 d에 대입된다.

<실행결과>

result = 1275.000000

Implicit type conversion (Coercion)

- 범위가 큰 데이터 형에서 작은 데이터 형으로의 변환은 문제 발생

형 변 환	문 제 점
큰 부동 소수점형을 작은 부동 소수점형으로 double ⇒ float	정밀도(유효 숫자)에 손실이 있다. 원래 값이 변환 데이터형의 범위를 벗어나는 경우의 결과는 예측 불허이다. ☞부동 소수점(floating point) = 실수(real number)
부동 소수[illegible]형을 정수형[illegible]로 float ⇒ [illegible]t	소수부를 잃어버리게 된다. 원래 값이 변화 데이터형의 범위를 벗어나는 경우는 결과는 예측 불허이다.
큰 정수[illegible]을 작은 정수[illegible]으로 long ⇒ [illegible]ort	원래 값이 변화 데이터형의 범위를 벗어나는 경우에 일반적으로 하위 바이트들만이 복사된다.

```
double x = 3458.234234;
float y
y = x;          // y = 3458.2341
```

Implicit type conversion (Coercion)

- 범위가 큰 데이터 형에서 작은 데이터 형으로의 변환은 문제 발생

형 변 환	문 제 점
큰 부동 소수점형을 작은 부동 소수점형으로 double ⇒ float	정밀도(유효 숫자)에 손실이 있다. 원래 값이 변환 데이터형의 범위를 벗어나는 경우의 결과는 예측 불허이다. ☞부동 소수점(floating point) = 실수(real number)
부동 소수점형을 정수형으로 float ⇒ int	소수부를 잃어버리게 된다. 원래 값이 변화 데이터형의 범위를 벗어나는 경우는 결과는 예측 불허이다.
큰 정[illegible]형을 작은 정[illegible]형으로 long [illegible] short	원래 값이 변화 데이터형의 범위를 벗어나는 경우에 일반적으로 하위 바이트들만이 복사된다.

```
float x = 3458.6543;
int y;
y = x;          // y = 3458
```

Implicit type conversion (Coercion)

- 범위가 큰 데이터 형에서 작은 데이터 형으로의 변환은 문제 발생

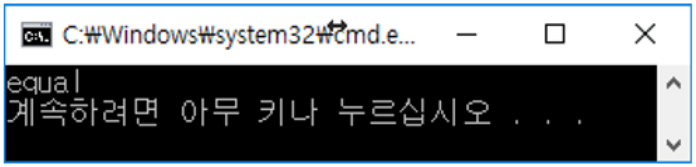
형 변 환	문 제 점
큰 부동 소수점형을 작은 부동 소수점형으로 double ⇒ float	정밀도(유효 숫자)에 손실이 있다. 원래 값이 변환 데이터형의 범위를 벗어나는 경우의 결과는 예측 불허이다. ☞부동 소수점(floating point) = 실수(real number)
부동 소수점형을 정수형으로 float ⇒ int	소수부를 잃어버리게 된다. 원래 값이 변화 데이터형의 범위를 벗어나는 경우는 결과는 예측 불허이다.
큰 정수형을 작은 정수형으로 long ⇒ short	원래 값이 변화 데이터형의 범위를 벗어나는 경우에 일반적으로 하위 바이트들만이 복사된다.

```
long x = 3458234;
short y;
y = x;          // y = −13108 (?)
```

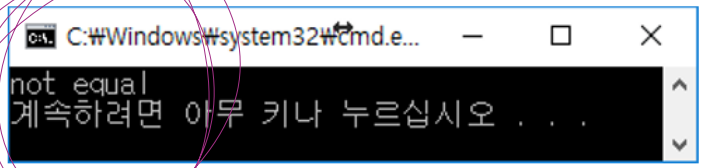
Implicit type conversion (Coercion)

◈ 자동 형변환의 예

```
#include <stdio.h>
void main(void) {
          float a = 0.5;
          if (a == 0.5) printf("equal\n");
          else printf("not equal\n");
}
```




```
#include <stdio.h>
void main(void) {
          float a = 0.2;
          if (a == 0.2) printf("equal\n");
          else printf("not equal\n");
}
```



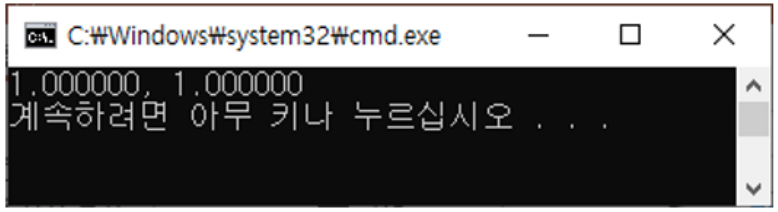
- 십진수 0.5는 이진수로 유한 소수지만 십진수 0.2는 이진수로 무한소수
- float형의 무한소수를 double형으로 형변환할 경우 double형의 무한소수와는 다른 이진 숫자가 된다.
- 소수에 대한 float형과 double형 사이의 형변환은 가능한 지양한다.
- 반드시 float형 변수에는 float형 상수를 사용하는 습관을 들인다.

Explicit type conversion (Cast)

◈ 강제 형변환 (Explicit type conversion, casting)

- 프로그램에서 강제로 형변환을 한다.
- 형변환 연산자 (cast operator)를 사용한다.
- 자동 형변환이 발생하더라도 가독성(readibility)을 위해서 가능하면 강제 형변환을 권장한다.
  - 아래의 코드에서 b = (float)(3/2); 만 보고 b의 데이터형을 유추할 수 있다.

```
#include <stdio.h>
void main(){
        float a, b;
        a = 3 / 2;                  // 3, 2는 int이므로 1
        b = (float)(3 / 2);         // float로 강제 형변환해서 1.0
        printf("%f, %f\n", a, b);
}
```



Explicit type conversion (Cast)

- **Explicit conversion (casting, cast operation)**
  - 프로그래머가 강제적으로 변수의 데이터형 변환 가능
  - 피연산자의 값을 앞 위치의 괄호 안에서 지정한 자료형으로 변환하는 연산

```
#include <stdio.h>

int main (void)
{
    double r1, r2;
    int result1, result2;

    r1 = 3.4;
    r2 = 2.1;

    result1 = r1 * r2;
    result2 = (int)r1 * (int)r2;

    printf("result1 = %d\n", result1);
    printf("result2 = %d\n", result2);

    return 0;
}
```

implicit type conversion

cast 연산자에 의한 explicit conversion

- 형태 : (type) expression
  - ➔ expression 의 값을 type의 데이터형으로 강제로 변환.

<실행결과>

result1 = 7 (원래는 7.14)
result2 = 6

Explicit type conversion (Cast)

- 시스템, OS (운영체제), 컴파일러의 종류에 따라 각 데이터형의 메모리 저장 크기가 다를 수 있음
  - int형 데이터가 2byte, 4byte, 심지어는 32byte인 컴퓨터도 있음
- 프로그램이 사용하는 데이터형의 크기를 알 수 있는 방법으로 sizeof 연산자를 사용할 수 있음 (unary operator)
  - sizeof (type)
    - 위의 연산자를 사용하면 해당 데이터형의 크기를 byte 단위로 알려줌

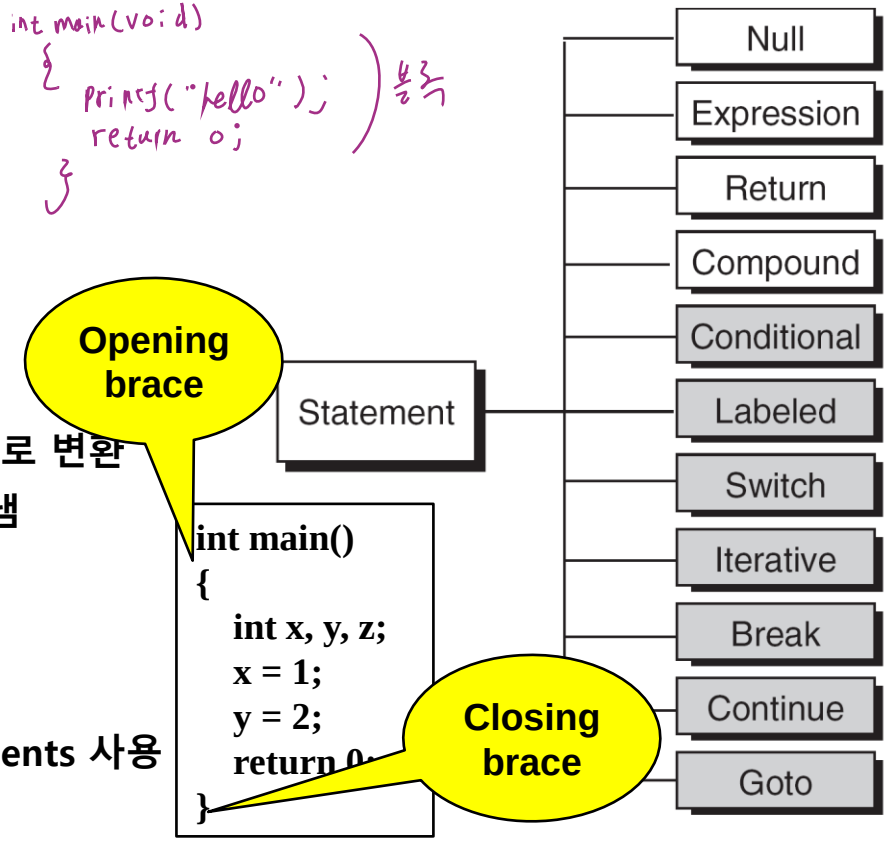
```
Ex)  sizeof(int) ;
     sizeof(float) ;
     sizeof -14.5 ;
```

cf) 시스템에 따라 데이터형의 크기가 다를 수 있다는 점은, 한 환경에서 다른 환경으로 프로그램을 가져갈 때 문제를 일으킬 수 있다.
sizeof() 연산자는 데이터형의 크기가 중요시 되는 프로그램에서 유용하게 사용될 수 있다.

Statements (명령문)

- 프로그램에 의해서 수행되는 하나의 동작(action)
- 직접적으로 하나 또는 그 이상의 실행 가능한 컴퓨터 명령어로 변환됨
- Statement의 종류
  - Null statement
    - ; ← statement가 있어야 하지만 아무런 동작도 필요하지 않을 경우 사용
    - loop등에서 유용하게 사용

      예) for(; ;)
  - Expression statements
    - 표현식 뒤에 세미콜론(;)을 붙여 statement로 변환
    - ';'는 컴퓨터에게 statement의 끝임을 나타냄
  - Compound statements
    - 0 (zero) 또는 그 이상의 문장으로 구성되는 코드 단위 : '{ ... }' 형태의 Block 단위
    - main 함수를 만들 때도 compound statements 사용
    - 세미콜론(;)이 필요하지 않음



세 정수의 합, 평균, 분산을 구하는 프로그램

```c
#include <stdio.h>

int main (void)
{
    int a, b, c;
    int sum;
    float mean;
    double variance;

    printf("Enter first integer:");
    scanf("%d", &a);
    printf("Enter first integer:");
    scanf("%d", &b);
    printf("Enter first integer:");
    scanf("%d", &c);

    sum = a + b + c;
    printf("Sum is : %d\n", sum);

    mean = (float)sum / 3;
    printf("Mean is : %f\n",mean);

    variance = ((a-mean)*(a-mean) + (b-mean)*(b-mean) + (c-mean)*(c-mean))/3.0;
    printf("Variance is : %lf\n", variance);

    return 0;
}
```

associativity, precedence, type conversion이 어떻게 적용되는지 살펴보자.

```
cse20161566@cspro:~$ gcc test.c
cse20161566@cspro:~$ ./a.out
Enter first integer: 1
Enter second integer: 2
Enter last integer: 34
Sum is :37
Mean is: 12.333333
Variance is:234.888901
cse20161566@cspro:~$
```

세 정수의 합, 평균, 분산을 구하는 프로그램

- **예) int sum = 2; 일 때, (mean은 float로 선언되었음)**
- **mean = (float) sum / 3.0;**
  - cast 연산자에 의한 explicit conversion : int -> float
  - 결과값 : mean = 0.666666...

  : 수행 후, sum의 값은? 2
- **mean= sum / 3.0;**
  - implicit type conversion : int -> float
  - 결과값 : mean = 0.666666...

  : 수행 후, sum의 값은? 2
- **mean= (float) sum / 3;**
  - cast 연산자에 의한 explicit conversion : int -> float
  - implicit type conversion : int -> float
  - 결과값 : mean = 0.666666...

  : 수행 후, sum의 값은? 2
- **mean = sum / 3;**
  - Implicit type conversion : int -> float
  - 결과값 : mean = 0.0

  : 수행 후, sum의 값은? 2

세 정수의 합, 평균, 분산을 구하는 프로그램

- **예) int sum = 2; 일 때, (계속)**
- **mean = (float) (sum / 3);**
  - cast 연산자에 의한 explicit conversion : int -> float
  - 결과값 : mean = 0.0

  : 수행 후, sum의 값은? 2
- **mean= sum / 3 * 100;**
  - implicit type conversion : int -> float
  - 결과값 : mean = 0.0

  : 수행 후, sum의 값은? 2
- **mean=100 * sum / 3;**
  - implicit type conversion : int -> float
  - 결과값 : mean = 66.0

  : 수행 후, sum의 값은? 2