한빛아카데미 Hanbit Academy, Inc.

C 언어 프로그래밍

Part 01. C 언어 준비 학습

Chapter 04. 표현식과 연산자

목차

학습목표

- **표현식과 명령문의 개념을 이해하고 종류를 살펴본다.**
- **연산자의 종류를 알아보고 사용해본다.**
- **여러 가지의 연산자가 등장했을 때 우선순위에 따라 연산할 수 있다.**

01

표현식과 명령문

01. 표현식과 명령문

I. 표현식(expression)

- **표현식의 개념**
  - 하나의 값으로 계산될 수 있는 것

```
표현식은 그 자체가 하나의 값이다.
```

- **표현식의 예시**
  - **[1], [3.141592], ['A']**
    C 언어에서 문자는 아스키코드나 유니코드로 하나의 값으로 여기기 때문에, 리터럴이 나타내는 값으로 계산됨
  - **[x], [y]**
    변수 x, y가 표현식일 경우 변수에 저장된 값으로 계산됨

01. 표현식과 명령문

I. 표현식(expression)

- **표현식의 예시**
  - **[(1 + 2) * 3], [(x + y) * 2]**
    연산자와 피연산자로 이루어진 표현식은 피연산자가 모두 리터럴과 변수임
  - **[x = 1], [x = y]**
    위와 같은 대입식의 계산값은 좌변에 우변이 대입된 후 좌변의 값이 됨
  - **[x < 1], [x > y], [x == 1], [x != y]**
    위와 같은 조건식은 참일 경우 1, 거짓일 경우 0으로 계산됨
  - **["난생처음"]**
    위와 같은 문자열 리터럴 자체가 하나의 값임

01. 표현식과 명령문

I. 표현식(expression)

- **표현식의 예시**
  - **[sum(1, 2)]**
    위와 같은 호출식은 반환 값으로 계산됨
  - **[y = sum(1, 2)]**
    호출식을 포함하는 대입식임

- **표현식의 응용**

```
[ (표현식1 + 표현식2) * 표현식3 ]
[ 표현식1 < 표현식2 ]
[sum(표현식1, 표현식2) ]
```

01. 표현식과 명령문

II. 명령문(statement)

- **명령문의 개념**
  - 계산을 비롯하여 컴퓨터에 무엇인가를 하도록 시키는 구문

- **명령문의 종류**
  - **선언문** : 이름(식별자)을 프로그램에 알려주는 명령문
  - **표현식문** : 표현식 끝에 세미콜론(;)을 붙여 실제 표현식이 값으로 계산되도록 만드는 명령문
  - **제어문** : 프로그램의 실행 흐름을 제어하는 명령문

01. 표현식과 명령문

II. 명령문(statement)

- **명령문의 예시**
  - **[1;]**
    표현식 끝에 세미콜론을 붙여 표현식문이 됨
  - **[int x;], [int x = 1;]**
    변수를 선언하며 세미콜론으로 끝나는 선언문
    정의문이라고도 할 수 있으며, 정의문은 선언문에 포함됨
  - **[x = 3;]**
    대입식 끝에 세미콜론을 붙여 표현식문이 됨

01. 표현식과 명령문

II. 명령문(statement)

- **명령문의 예시**
  - **[Add(1, 2);]**
    호출식 끝에 세미콜론을 붙여 표현식문이 됨
    호출문이라고도 함
  - **[y = Add(1, 2);]**
    호출식을 포함하는 대입식에 세미콜론을 붙여 표현식문이 됨
  - **[{ int a; a = 1 + 2; }]**
    하나 이상의 명령문을 중괄호로 감싸 블록으로 만들 경우
    하나의 명령문으로 취급되며 복합문이라고 함
    블록 끝에는 세미콜론을 붙이지 않음

01. 표현식과 명령문

확인문제1

[int x; x = 1;]과 [{ int x; x = 1; }]의 차이를 설명하시오.

01. 표현식과 명령문

LAB 4-1 직육면체의 부피 구하기

다음 출력처럼 직육면체의 가로, 세로, 높이를 공백으로 구분하여 입력받고 곱셈 함수 `Multiply(int a, int b)`만을 호출하여 부피를 구한 후 출력해봅시다.

```
직육면체의 가로, 세로, 높이를 입력하세요.
2 4 8
부피: 64
```

1. 가로, 세로, 높이에 해당하는 변수를 정의한 후에 입력받는다.
2. '가로*세로'를 구한 계산 값과 높이의 곱을 `Multiply`를 중첩 호출하여 구한다.

01. 표현식과 명령문

LAB 4-1 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int Multiply(int a, int b)
05  {
06      return a * b;
07  }
08
09  int main()
10  {
11      int width, depth, height;
12
13      printf("직육면체의 가로, 세로, 높이를 입력하세요.\r\n");
14      scanf("%d%d%d", &width, &depth, &height);
15
16      int volume = Multiply(Multiply(width, depth), height);
17
18      printf("부피: %d", volume);
19  }
```

02

연산자란?

02. 연산자란?

I. 연산자와 피연산자의 개념

- **연산자**
  - 더하기(+), 빼기(-), 곱하기(*), 나누기(/)
  - 대입(=), 상등(==), 부등(!=)

- **피연산자**
  - 연산자의 대상이 되는 것

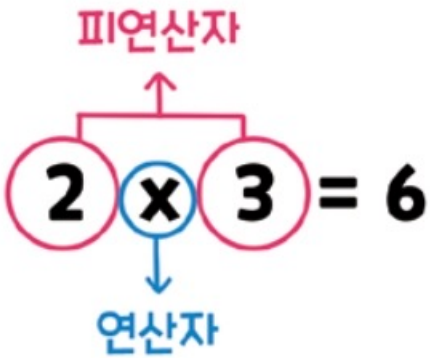


그림 4-1 연산자와 피연산자

02. 연산자란?

II. 연산자의 분류

- **기능별 분류**
  - 산술 연산자, 비교 연산자, 논리 연산자, 비트 연산자

- **형태별 분류**
  - **단항 연산자** : 피연산자가 1개인 연산자
  - **이항 연산자** : 피연산자가 2개인 연산자
  - **삼항 연산자** : 피연산자가 3개인 연산자

03

산술 연산자

03. 산술 연산자

I. 사칙 연산자

- **사칙 연산자의 형태**

```
a op b (단, op는 +, -, *, /이다.)
→ 연산 결과는 a와 b의 타입에 따라 달라질 수 있다.
```

- **사칙 연산자의 원칙**
  - 두 피연산자의 타입이 동일해야 함
  - 계산 값 역시 같은 타입이 됨

03. 산술 연산자

II. 대입 연산자

- **대입 연산자의 형태**

```
a = b
→ a에 b를 대입한다.
→ 연산 결과는 a이다.
```

- **대입 연산자의 원칙**
  - 두 피연산자의 타입이 동일해야 함
  - 동일하지 않을 경우, 컴파일러는 좌측 피연산자의 타입의 변환을 수행함
  - ex. `int a = 3.7` → 정수형 = 실수형
    우측 피연산자 3.7을 좌측 피연산자 타입으로 변환하여 임시로 정수형 3으로 만든 후 대입하면 변수 a는 정수형 3을 가짐

03. 산술 연산자

II. 대입 연산자

- **대입 연산자의 원칙**
  - 대입 연산자가 피연산자를 사이에 둔 채 연달아 나올 경우, 오른쪽 대입 연산자가 먼저 적용됨

```
int a, b;
a = b = 1;
```

[코드 4-1] 대입 연산자

```
01  #include <stdio.h>
02
03  int main()
04  {
05      int a, b;
06      printf("%d\r\n", a = b = 1);
07      printf("%d%d", a, b);
08  }
```

```
1
1 1
```

03. 산술 연산자

III. 나머지 연산자

- **나머지 연산자(Modular)의 형태**
  - 나머지를 구하는 연산자

```
a % b (단, a, b는 정수이다.)
→ a를 b로 나머지 있는 나눗셈을 한다.
→ 연산 결과는 나머지이다.
```

03. 산술 연산자

III. 나머지 연산자

[코드 4-2] 2의 배수 판별

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main()
05  {
06      int a;
07      printf("숫자를 입력하세요.\r\n");
08      printf("2의 배수면 1, 아니면 0을 출력\r\n");
09      scanf("%d", &a);
10      printf("결과: %d", a % 2 == 0);
11  }
```

```
숫자를 입력하세요.
2의 배수면 1, 아니면 0을 출력
8
결과: 1
```

IV. 증감 연산자

- **증감 연산자의 개념**
  - 1을 증감하는 식을 축약할 수 있는 연산자(++, --)

```
a++ → 연산 결과는 a이다. 연산 후 a는 1 증가한다.
++a → 연산 결과는 a를 1 증가시킨 후의 a이다.
a-- → 연산 결과는 a이다. 연산 후 a는 1 감소한다.
--a → 연산 결과는 a를 1 감소시킨 후의 a이다.
```

  - 단항 연산자로서 피연산자의 앞(전위형)이나 뒤(후위형)에 올 수 있음

```
전위형 ++a, --a → 증감 연산자가 피연산자 앞에 있다.
후위형 a++, a-- → 증감 연산자가 피연산자 뒤에 있다.
```

03. 산술 연산자

IV. 증감 연산자

[코드 4-3] 전위형과 후위형의 차이 1

```
01  #include <stdio.h>
02
03  int main()
04  {
05      int a;
06
07      a = 1;
08      printf("++a: %d,\r\n", ++a);
09
10      a = 1;
11      printf("a++: %d", a++);
12  }
```

전위형

후위형

```
++a: 2
a++: 1
```

IV. 증감 연산자

[코드 4-3] 전위형과 후위형의 차이 2

```
01  #include <stdio.h>
02
03  int main()
04  {
05      int a, b;
06
07      a = 1;
08      b = ++a;
09      printf("a: %d, b: %d\r\n", a, b);
10
11      a = 1;
12      b = a++;
13      printf("a: %d", b: %d", a, b);
14  }
```

전위형

후위형

```
a: 2, b: 2
a: 2, b: 1
```

03. 산술 연산자

확인문제3

1. 프로그래밍에서 7 / 2 * 2의 계산 값을 구하시오.

2. 다음 명령문을 실행할 때 d의 값을 구하시오.

```
int a;
double d = a = 3.7;
```

3. 다음 식을 계산하시오.

   (1) -7 % 2

   (2) -7 % -2

4. 두 정수 변수 a, b에 대해서 a = b = 1인 경우, a++ + ++b를 계산하시오.

03. 산술 연산자

LAB 4-2 **세 자리 자연수의 각 자리 수 구하기**

다음 실행 결과를 참고하여 세 자리 자연수를 입력받은 후 각 자리의 수를 출력하는 프로그램을 작성해봅시다.

```
세 자리 자연수를 입력하세요.
345
3, 4, 5
```

1. 입력받는 변수 a와 각 자리의 수를 저장하는 변수 r1, r2, r3을 정의한다.
2. a/100은 a와 100이 모두 정수형이므로 백의 자리 정수로 계산된다.
3. (a % 100)은 a에서 백 자리 이상을 날려버린 수이고 10으로 나누면 역시 십의 자리 정수로 계산된다.
4. (a % 10)은 a에서 십 자리 이상을 날려버린 수이므로 일의 자릿수를 나타낸다.

03. 산술 연산자

LAB 4-2 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main()
05  {
06      int a, r1, r2, r3;
07
08      printf("세 자리 자연수를 입력하세요.\r\n");
09      scanf("%d", &a);
10
11      r1 = a / 100;
12      r2 = (a % 100) / 10;
13      r3 = (a % 10);
14
15      printf("%d, %d, %d", r1, r2, r3);
16  }
```

04

비교 연산자

04. 비교 연산자

I. 비교 연산자

- **비교 연산자의 개념**
  - 왼쪽 피연산자가 오른쪽 피연산자와 비교하여 [같다, 같지 않다, 크다, 작다, 크거나 같다, 작거나 같다]를 평가하는 데 사용함

- **비교 연산자의 형식**

```
a op b (단, op는 ==, !=, >, <, >=, <=이다.)
→ a를 b와 비교한다.
→ 연산 결과는 참이면 1, 거짓이면 0이다.
```

04. 비교 연산자

II. 비교 연산의 결과

- 비교 연산자의 계산 결과는 1(참), 0(거짓)으로 나타냄

표 4-1 비교 연산자

	a와 b가 같다	a가 b보다 크다	a가 b보다 작다
a == b	1	0	0
a != b	0	1	1
a > b	0	1	0
a < b	0	0	1
a >= b	1	1	0
a <= b	1	0	1

04. 비교 연산자

II. 비교 연산의 결과

[코드 4-5] 조건식의 값

```
01 #include <stdio.h>
02
03 int main()
04 {
05     int c1 = (1 == 1);    // 참
06     int c2 = (1 != 1);    // 거짓
07     int c3 = (2 > 1);     // 참
08     int c4 = (2 < 1);     // 거짓
09     int c5 = (2 >= 1);    // 참
10     int c6 = (2 <= 1);    // 거짓
11
12     printf("%d %d %d %d %d %d", c1, c2, c3, c4, c5, c6);
13 }
```

```
1 0 1 0 1 0
```

04. 비교 연산자

II. 비교 연산의 결과

[코드 4-6] 비교 연산자

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main()
05  {
06      int age;
07      printf("나이를 입력하세요.\r\n");
08      printf("미성년자면 0, 성인이면 1을 출력\r\n");
09      scanf("%d", &age);
10      printf("결과: %d", age >= 20);
11  }
```

```
나이를 입력하세요.
미성년자면 0, 성인이면 1을 출력
20
결과: 1
```

04. 비교 연산자

확인문제4

조건식 `((2 > 1) < 0) != 1`을 계산하시오.

04. 비교 연산자

LAB 4-3 **3의 배수 판별하기**

다음 출력처럼 입력받은 수가 3의 배수인 경우 1, 아닌 경우 0을 출력하는 프로그램을 작성해봅시다.

```
정수를 입력하세요.
15
1
```

n % 3 == 0은 n이 3의 배수일 때 1, 아닐 때 0으로 계산된다.

04. 비교 연산자

LAB 4-3 정답

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03
04 int main()
05 {
06    int n;
07    printf("정수를 입력하세요.\r\n");
08    scanf("%d". &n);
09    printf("%d", n % 3 == 0);
10 }
```

05

논리 연산자

05. 논리 연산자

I. 논리 연산자의 개념

- 피연산자에 적용되어 연산 결과가 1(참) 또는 0(거짓)이 되는 연산자
- 논리 부정(!), 논리곱(&&), 논리합(||)이 있음

II. 논리 부정 연산자

- **논리 부정 연산자의 형태**

```
!a
→ 연산 결과는 a의 논리 부정 값이다. 연산 후 a는 변하지 않는다.
```

- **논리 부정 연산의 결과**

표 4-2 논리 부정 연산

a	!a
0	1
0 아님	0

05. 논리 연산자

II. 논리 부정 연산자

- **논리 부정 연산자의 주의할 점**

  1) 이중 부정의 결과값 : a와 !!a는 동치가 아님

[코드 4-7] 이중 부정

```
01  #include <stdio.h>
02
03  int main()
04  {
05      int a = 2;
06      printf("a:%d\r\n!!a:%d", a, !!a);
07  }
```

```
a:2
!!a:1
```

05. 논리 연산자

II. 논리 부정 연산자

- **논리 부정 연산자의 주의할 점**

2) 논리 부정 연산자의 피연산자는 정수형이어야 함

3) 전위형만 존재함

4) 피연산자에 영향을 끼치지 않음

05. 논리 연산자

III. 논리곱 연산자

- 논리곱 연산자의 형태

```
a && b
→ 연산 결과는 a와 b 둘 다 0이 아닐 때 1, 둘 중 하나라도 0이면 0이다.
```

- 논리곱 연산의 결과

표 4-3 논리곱 연산

a	b	a && b
0	0	0
0	0 아님	0
0 아님	0	0
0 아님	0 아님	1

05. 논리 연산자

III. 논리곱 연산자

- **논리곱 연산자의 특징**
  - **&& Short-Circuit** : 논리식 [a && b]는 a가 만일 0이라면 b의 값에 상관없이 항상 0으로 계산됨

[코드 4-8] AND Short-Circuit

```
01  #include <stdio.h>
02
03  int main()
04  {
05      0 && printf("0");
06      1 && printf("1");
07  }
```

```
1
```

05. 논리 연산자

IV. 논리합 연산자

- 논리합 연산자의 형태

```
a || b
→ 연산 결과는 a와 b 둘 다 0일 때 0, 아니면 1이다.
```

- 논리합 연산의 결과

표 4-4 논리합

a	b	a || b
0	0	0
0	0 아님	1
0 아님	0	1
0 아님	0 아님	1

05. 논리 연산자

IV. 논리합 연산자

- **논리합 연산자의 특징**
  - **|| Short-Circuit** : 논리식 [a || b]는 a가 0이 아니면 b의 값에 상관없이 항상 1이 되기 때문에 b의 값을 구하지 않음

[코드 4-9] OR Short-Circuit

```
01  #include <stdio.h>
02
03  int main()
04  {
05      0 || printf("0");
06      1 || printf("1");
07  }
```

```
0
```

05. 논리 연산자

확인문제5

1. `double d = 3.141592`일 때, `!!d`를 계산하시오.

2. 함수 `f()`의 반환 값이 `0`인 경우만 함수 `g()`를 호출하는 명령문을 작성하시오.

   (1) `&&` Short-Circuit 사용 : _____

   (2) `||` Short-Circuit 사용 : _____

05. 논리 연산자

LAB 4-4 **0 판별하기**

다음 출력처럼 입력받는 수를 평가하여 0인지 아닌지를 판별하는 프로그램을 작성해봅시다.

```
정수를 입력하세요.
5
0이 아닙니다.
```

1. || Short-Circuit으로 인하여 n이 0일 때만 printf가 호출된다.
2. && Short-Circuit으로 인하여 n이 0이 아닐 때만 printf가 호출된다.

05. 논리 연산자

LAB 4-4 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main()
05  {
06      int n;
07      printf("정수를 입력하세요.\r\n");
08
09      scanf("%d", &n);
10      n || printf("0입니다.");
11      n && printf("0이 아닙니다.");
12  }
```

06

비트 연산자

06. 비트 연산자

I. 비트 연산자의 개념

- 프로그래밍에서만 지원되는 연산자
- 피연산자는 정수만 가능함
- 피연산자의 모든 비트에 대해서 개별적으로 연산이 적용됨
- 비트 연산 후에도 피연산자는 변하지 않음

06. 비트 연산자

II. 비트 반전 연산자

- 비트 반전(NOT) 연산자의 형태

~a
→ 연산 결과는 a의 모든 비트를 반전한 값이다. 연산 후에도 a는 변하지 않는다.

[코드 4-10] 비트 반전 연산자

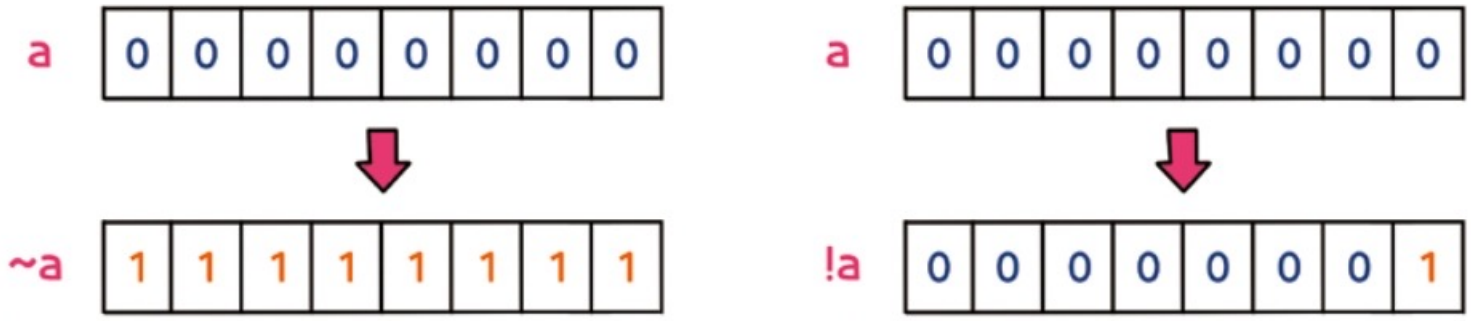
```
01  #include <stdio.h>
02
03  int main()
04  {
05      char a = 0;
06      printf("a:%d\r\n~a:%d", a, ~a);
07  }
```

```
a:0
~a:-1
```

06. 비트 연산자

II. 비트 반전 연산자

- **비트 반전 연산자와 논리 부정 연산자의 차이점**
  - 비트 연산자는 피연산자의 모든 비트에 대해 적용됨



(a) 비트 반전 연산 (b) 논리 부정 연산

그림 4-2 비트 반전(NOT) 연산자와 논리 부정(NOT) 연산자의 차이점

06. 비트 연산자

III. 비트곱, 비트합, 배타적 비트합 연산자

- **비트곱, 비트합, 배타적 비트합 연산의 결과**
  - a, b는 비트 연산자의 피연산자가 아닌 피연산자의 개별적인 하나의 비트 값

표 4-5 비트곱, 비트합, 배타적 비트합

a	b	a & b	a ¦ b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

06. 비트 연산자

III. 비트곱, 비트합, 배타적 비트합 연산자

- 배타적 비트합(XOR) 수행 결과

[코드 4-11] 같은 변수의 XOR

```
01  #include <stdio.h>
02
03  int main()
04  {
05      int a = 7;
06      printf("a XOR a: %d", a ^ a);
07  }
```

```
a ^ a: 0
```

06. 비트 연산자

IV. 비트 이동 연산자

- **비트 이동 연산자의 개념**
  - 정수 타입 값의 모든 비트를 왼쪽이나 오른쪽으로 이동시키는 연산자
- **비트 이동 연산자의 형태**

```
a << N → 연산 결과는 a의 각 비트를 왼쪽으로 N칸씩 이동시킨 새로운 값이다.
a >> N → 연산 결과는 a의 각 비트를 오른쪽으로 N칸씩 이동시킨 새로운 값이다.
→ 비트 이동 후의 빈자리는 <<은 0으로, >>은 0 혹은 1로 채운다. a는 연산 후에도 변하지 않는다.
```

06. 비트 연산자

IV. 비트 이동 연산자

- **왼쪽 비트 이동 연산 (<<)**
  - 새로 생성되는 비트는 0으로 채워짐

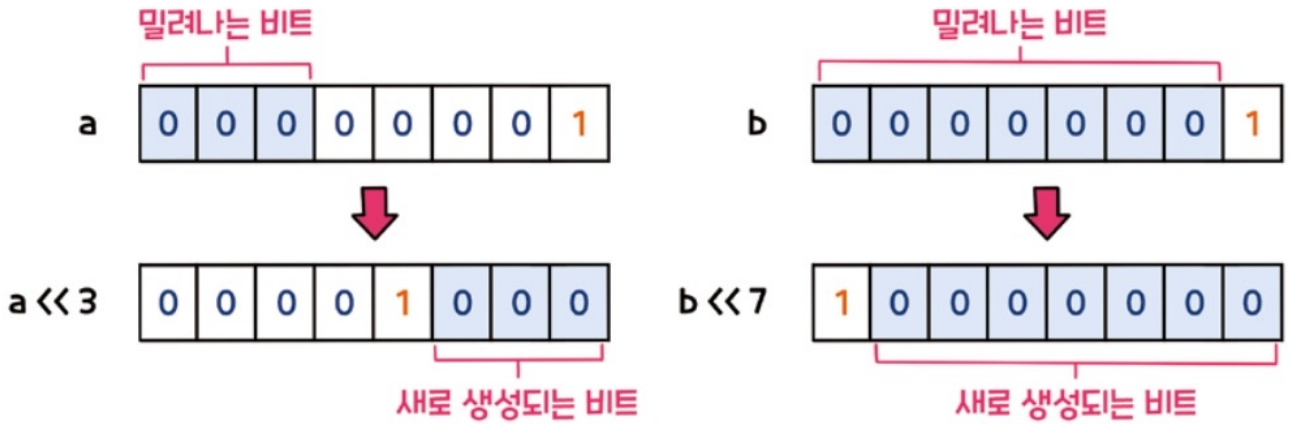


그림 4-3 정수 1의 왼쪽 비트 이동 연산

06. 비트 연산자

IV. 비트 이동 연산자

- 왼쪽 비트 연산

[코드 4-12] 왼쪽 비트 이동

```
01  #include <stdio.h>
02
03  int main()
04  {
05      unsigned char a = 0b1;
06      printf("a << 3: %d\r\n", a << 3);
07
08      unsigned char b = 0b1;
09      printf("b << 7: %d", b << 7);
10  }
```

```
a << 3: 8
b << 7: 128
```

06. 비트 연산자

IV. 비트 이동 연산자

- **오른쪽 비트 이동 연산 (>>)**
  - 부호 없는 정수의 경우 : 새로 생성되는 비트는 0으로 채워짐

그림 4-4 오른쪽 비트 이동 연산(부호 없는 정수형)

06. 비트 연산자

IV. 비트 이동 연산자

- **오른쪽 비트 이동 연산 (>>)**
  - 부호 있는 정수의 경우 : 부호 비트(가장 왼쪽 비트)에 있는 값이 새로 생성되는 비트로 채워짐

그림 4-5 오른쪽 비트 이동 연산(부호 있는 정수형)

06. 비트 연산자

IV. 비트 이동 연산자

- 오른쪽 비트 이동 연산 (>>)

[코드 4-13] 오른쪽 비트 이동

```
01  #include <stdio.h>
02
03  int main()
04  {
05      unsigned char uc = 0b10000000;
06      printf("uc >> 7: %d\r\n", uc >> 7);
07
08      char c = 0b1;
09      printf("c >> 7: %d", c >> 7);
10  }
```

```
uc >> 7: 1
c >> 7: -1
```

부호 있는 정수형 변수의 모든 비트가 1로 채워진 경우 항상 –1을 나타냅니다.

06. 비트 연산자

확인문제6

1. ~-1의 값을 계산하시오.

2. 0b10000001과 0b01111110의 배타적 비트 합을 계산하시오.

3. 변수 a에 대해서 a * 7을 더하기와 비트 이동 연산자만을 사용해서 나타내시오.

06. 비트 연산자

LAB 4-5 **세 번째 비트 값 구하기**

입력받은 숫자의 세 번째 비트 값을 출력하는 프로그램을 작성해봅시다.

```
정수를 입력하세요.
5
세 번째 비트 : 1
```

1. `0b100`은 세 번째 비트만 1인 정수이다. 따라서 `(n & 0b100)`은 n의 세 번째 비트만 그대로 두고 나머지 비트는 모두 0인 값이 된다.
2. 해당 값을 2칸 오른쪽 비트 이동할 경우 0 혹은 1이 된다.

06. 비트 연산자

LAB 4-5 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main()
05  {
06      int n;
07      printf("정수를 입력하세요.\r\n");
08
09      scanf("%d", &n);
10      printf("세 번째 비트: %d", (n & 0b100) >> 2);
11  }
```

07

복합 대입 연산자와 삼항 연산자

07. 복합 대입 연산자와 삼항 연산자

I. 복합 대입 연산자

- **복합 대입 연산자의 형태와 의미**

```
a op= b
→ 연산 결과는 a = a op b를 계산한 값과 같다.
```

표 4-6 복합 대입 연산자

연산자	의미
a += b	a = a + b
a -= b	a = a - b
a *= b	a = a * b
a /= b	a = a / b
a %= b	a = a % b
a &= b	a = a & b
a |= b	a = a | b
a ^= b	a = a ^ b
a <<= b	a = a << b
a >>= b	a = a >> b

07. 복합 대입 연산자와 삼항 연산자

II. 삼항 연산자

- 삼항 연산자의 형태

조건식 ? 표현식1 : 표현식2
→ 연산 결과는 조건식이 참일 경우 표현식1, 거짓일 경우 표현식2를 계산한다.

[코드 4-14] 삼항 연산자

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main()
05  {
06      int age;
07      printf("나이를 입력하세요.\r\n");
08      scanf("%d", &age);
09      printf("%s", age < 20 ? "미성년자" : "성인");
10  }
```

```
나이를 입력하세요.
18
미성년자
```

07. 복합 대입 연산자와 삼항 연산자

확인문제7

1. a가 4일 때, 다음 연산을 수행한 후 a의 값은 무엇인가?

```
a += 5;
```

2. 함수 f()의 반환 값이 0일 때는 함수 g()를 호출하고, 0이 아닐 때는 함수 h()를 호출하는 명령문을 작성하시오.

07. 복합 대입 연산자와 삼항 연산자

LAB 4-6 **입력받은 수보다 큰 다음 짝수 구하기**

입력받은 수보다 큰 다음 짝수를 출력하는 프로그램을 작성해봅시다. 예를 들어 5를 입력할 경우 6이 출력되고, 6을 입력할 경우 8이 출력됩니다.

```
정수를 입력하세요.
6
다음 짝수 : 8
```

조건식 n % 2 == 0이 참이면 n이 짝수이므로 n에 2를 더하고, 거짓이면 n이 홀수이므로 n에 1을 더한다.

07. 복합 대입 연산자와 삼항 연산자

LAB 4-6 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main()
05  {
06      int n;
07      printf("정수를 입력하세요.\r\n");
08
09      scanf("%d", &n);
10      n += n % 2 == 0 ? 2 : 1;
11      printf("다음 짝수: %d", n);
12  }
```

08

기타 연산자와 우선순위

08. 기타 연산자와 우선순위

I. sizeof 연산자

- **sizeof 연산자의 개념**
  - 피연산자가 차지할 수 있는 메모리 영역의 크기를 바이트 단위로 나타냄
- **sizeof 연산자의 형태**

❶ `sizeof(타입 or 표현식)`

❷ `sizeof 표현식`

→ 연산 결과는 타입 크기 or 표현식이 차지하는 메모리 영역의 크기(바이트 단위)이다.

08. 기타 연산자와 우선순위

I. sizeof 연산자

- sizeof 연산자의 사용 방식

[코드 4-15] sizeof

```
01  int main()
02  {
03      int a;
04
05      int s1 = sizeof(a);
06      int s2 = sizeof a;
07      int s3 = sizeof(int);
08      int s4 = sizeof int;
09      int s5 = sizeof(9999);
10      int s6 = sizeof 9999;
11  }
```

오류 – 표현식이 아님

08. 기타 연산자와 우선순위

II. 타입 변환 연산자

- **타입 변환 연산자의 개념**
  - 계산된 값에 타입 변환 연산자(명시적 타입 변환)를 적용할 경우 변환 타입에 맞는 임시 값이 도출됨
- **타입 변환 연산자의 형태**

```
(타입)표현식
→ 연산 결과는 '타입'으로 변환된 표현식의 계산 값(임시 값)이다.
```

08. 기타 연산자와 우선순위

II. 타입 변환 연산자

- **암시적 타입 변환**
  - 타입 변환 연산자가 사용되지 않고, 자동으로 타입 변환이 됨

[코드 4-17] 산술 연산의 암시적 타입 변환

```
01 #include <stdio.h>
02
03 int main()
04 {
05     int a = 33554434;
06     float b = 0.1;
07
08     int c1 = a + b;
09     int c2 = a + (int)b;
10
11     printf("c1: %d, c2: %d", c1, c2);
12 }
```

```
c1: 33554432, c2: 33554434
```

08. 기타 연산자와 우선순위

II. 타입 변환 연산자

- **암시적 타입 변환과 명시적 타입 변환 비교**
  - 타입 변환을 할 경우 값을 대입받을 수 없음

[코드 4-18] 암시적 타입 변환과 명시적 타입 변환

```
01  int main()
02  {
03      char c = 0;
04
05      c = 1;
06      (int)c = 2;
07      (char)c = 3;
08  }
```

오류 발생

논쟁의 여지

08. 기타 연산자와 우선순위

III. 연산자의 우선순위

- **결합법칙**
  - 연산자들이 같은 우선순위에 있을 경우 적용 순서
  - 연산 순서에 따라 값이 달라지기 때문에 괄호를 사용해야 함

그림 4-6 연산자의 우선순위에 따라 달라지는 결과 값

08. 기타 연산자와 우선순위

III. 연산자의 우선순위

- **결합법칙**

표 4-7 연산자의 우선순위와 결합법칙

순위	연산자	결합법칙
1	(), [], ->, .(직접 멤버)	왼쪽에서 오른쪽
2	++, --, +(부호), -(부호), !, ~, &(참조), *(간접), sizeof, (TYPE)	오른쪽에서 왼쪽
3	*(곱하기), /, %	왼쪽에서 오른쪽
4	+(더하기), -(빼기)	왼쪽에서 오른쪽
5	<<, >>	왼쪽에서 오른쪽
6	<, <=, >=, >	왼쪽에서 오른쪽
7	==, !=	왼쪽에서 오른쪽
8	&(비트)	왼쪽에서 오른쪽
9	^	왼쪽에서 오른쪽
10	|	왼쪽에서 오른쪽
11	&&	왼쪽에서 오른쪽
12	||	왼쪽에서 오른쪽
13	?:(삼항)	오른쪽에서 왼쪽
14	=, +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=	오른쪽에서 왼쪽

08. 기타 연산자와 우선순위

확인문제8

1. `sizeof(0.0)`의 결과는 무엇인가?

2. `char c = -1;`일 때, `(unsigned char)c`를 계산하시오.

3. 연산자 우선순위 표를 참고하여 `1 || 0 && 0`을 계산하시오.

08. 기타 연산자와 우선순위

LAB 4-7 가우스 함수 f(x) = [x] 구현하기

실수 x에 대하여 x를 넘지 않는 최대의 정수를 반환하는 가우스 함수를 구현하고, 입력을 받아서 함수값을 출력하는 프로그램을 작성해봅시다.

```
실수를 입력하세요.
3.15
Gaus : 3
```

힌트

1. 실수 x가 0 이상이거나 정수인 경우 가우스 함수 값은 (int)x이다.
2. 실수 x가 0보다 작고 정수가 아닌 경우 가우스 함수 값은 (int)x-1이다.
3. (int)x를 g라고 할 때 ❷를 정리하면 x – g < 0이다. 이것을 삼항 연산자를 이용하여 정리하면 가우스 함수의 반환 값 g = x – g < 0 ? > g – 1 : g로 정리된다.

08. 기타 연산자와 우선순위

LAB 4-7 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int Gaus(double x)
05  {
06      int g = (int)x;
07      g = x - g < 0 ? g - 1 : g;
08      return g;
09  }
10
11  int main()
12  {
13      double d;
14      printf("실수를 입력하세요.\r\n");
15
16      scanf("%lf", &d);
17      printf("Gaus: %d", Gaus(d));
18  }
```

[실전예제]

두 수의 대소 관계 파악하기

[실전예제] 두 수의 대소 관계 파악하기

[문제]

두 수를 입력하면 두 수의 대소 관계를 파악하여 표현하는 프로그램을 작성해봅시다.

실행 결과

```
두 정수를 공백으로 구분하여 입력하세요.
9 3
9 > 3
```

[실전예제] 두 수의 대소 관계 파악하기

[해결]

1. 두 매개변수 x, y에 대해서 x<y이면 -1, x==y이면 0, x>y이면 1을 반환하는 compare 함수를 만든다.

2. 반환 값은 삼항 연산자를 중첩 사용해서 구한다.

3. 입력받은 두 정수를 compare에 전달하여 대소 관계를 확인한다.

4. printf의 서식 문자열을 삼항 연산자를 중첩 사용하여 결정한다.

[실전예제] 두 수의 대소 관계 파악하기

[해결]

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int compare(int x, int y)
05  {
06      int r = x - y < 0 ? -1 : x == y ? 0 : 1;
07      return r;
08  }
09
10  int main( )
11  {
12     int x, y;
13     printf("두 정수를 공백으로 구분하여 입력하세요.\r\n");
14
15     scanf("%d%d", &x, &y);
16
17     int c = compare(x, y);
18     printf(c < 0 ? "%d < %d" : c == 0 ? "%d == %d" : "%d > %d", x, y);
19  }
```

Thank you!

한빛아카데미 Hanbit Academy, Inc.