3 연산자

3-1. 연산자란?

3-2. 연산자의 종류

3-2-1. 대입 연산자

3-2-2. 산술 연산자

3-2-3. 복합 대입 연산자

3-2-4. 증감 연산자

3-2-5. 비교 연산자

3-2-6. 논리 연산자

3-2-7. 기타 연산자

@심화: 비트 연산자

3-3. 연산자 우선순위

3-1. 연산자란?

두 가지 비유를 통하여 연산자를 소개하겠다. 수학 문제를 풀 때를 떠올려보면 그 과정은 언제나 숫자와 기호들을 통해 연산하여 값을 구하는 것이었다. 프로그래밍에서의 연산도 수학 문제와 같다. “변수”를 “연산자”라는 기호를 통하여 연산해 “값”을 뱉어내는 것이다. 이러한 연산에서 값을 가지는 것을 수식이라 부른다. 아래의 그림을 통하여 수식이라 불리는 것들을 살펴보자.

(그림: 2-1, 조금 큰 아이의 모습)

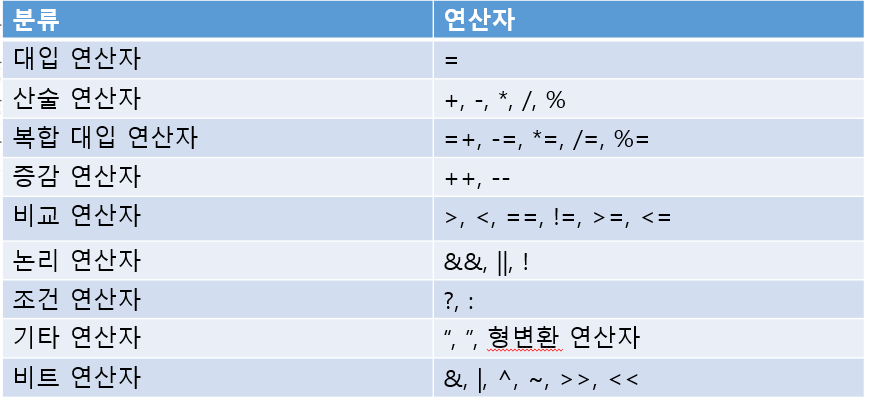
앞 단원의 비유를 이어서 설명해보겠다. 하나의 가구를 만드는 과정에 비유하자면, 연산자란 결국 그릇에 담은 재료를 가공하는 도구들이다. 가구를 만들기 위해서 컴퓨터는 재료를 설계도에 적힌 도구를 이용해 순서대로 가공해야 한다. 연산자는 톱과 망치 등에 해당하는 아주 기본적인 도구이다.



연산자에 대한 자세한 설명은 위의 예시를 통하여 설명하겠다. 그림과 같이 연산자가 들어있는 수식을 연산식이라 부르는데, 연산자에게 연산을 당하는 저 친구들을 우리는 “피연산자”라고 부른다. 피연산자의 개수에 따라 단항 연산자, 이항 연산자 등으로 불린다. 아래의 예시를 참고하자.



뿐만 아니라, 연산자는 쓰이는 기능에 따라 분류되기도 한다. 산술 연산자, 증감 연산자, 관계 연산자 등 종류가 많다. 아래의 표를 통하여 참고하자.

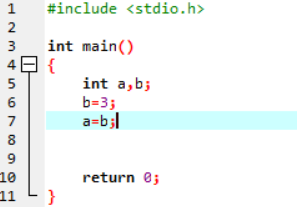


종류가 많지만 이해만 하면 쉬운 내용이니, 지금부터 위 표에 제시된 연산자를 하나하나 자세히 알아가도록 하자.

3-2. 연산자의 종류

3-2-1. 대입 연산자

가장 기본이 되는 연산자이다. 기본적인 대입 연산자는 “=”으로, 수학에서는 좌변의 값과 우변의 값이 같다는 의미로 쓰였지만, 프로그래밍에서는 조금 다르다. 수학의 연산 기호와는 달리 C언어에서의 “=”은 우변의 “값”을 좌측의 “변수”에 대입한다는 의미이다.

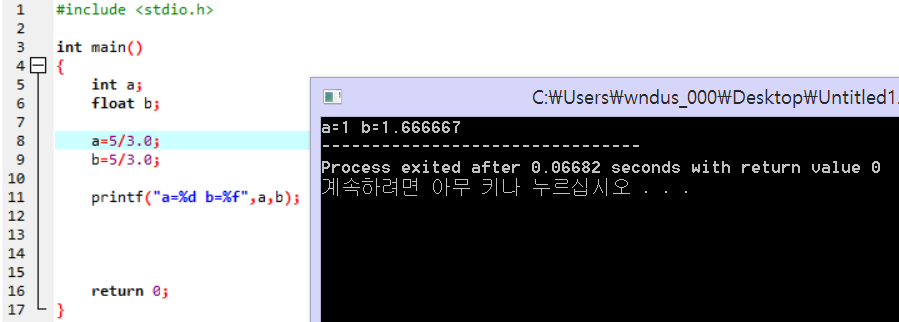


다른 연산자와 혼합하여 더욱 심화된 대입 연산자도 있으나, 그것은 다른 연산자를 학습한 뒤에 소개하겠다. (\*) 조금 뒤의 복합 대입 연산자 단원에서 심화된 내용을 배우도록 하자.

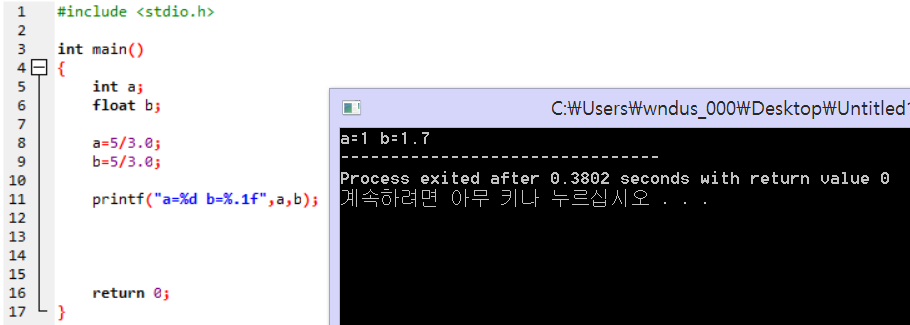
3-2-2. 산술 연산자

산술 연산자는 기본적인 사칙 연산을 제공하는 연산자이다. 대부분 수학과 같은 의미를 가지는 연산자들 이지만 두 가지 예외인 연산자가 있다.

“/” 연산자의 경우 항상 몫을 구하는 연산자이다. 그러나 범위가 정수가 아닌 실수로 넘어가면(즉 변수가 선언을 int가 아닌 float으로 할 경우), 그 몫 역시 실수 이므로 정수에서는 나머지로 무시되던 부분까지 전부 몫으로 표현이 된다.



연산자의 입장에서 봤을 때에는 정수끼리의 계산이니 결과 역시 정수로 표현해 주는 것이 당연하고, 실수끼리의 계산 역시 실수로 표현해 주는 것이 당연하다. 하지만 이런 경우 나머지가 float 소수점 아래 5자리까지 표시되므로 소수점 아래 한 자리만 궁금한 경우라면 앞의 2단원에서 배운 방법을 사용하여 “%.1f”으로 제한해 주는 것이 바람직하다.



3-2-3. 복합 대입 연산자

이제 대입연산자의 심화된 종류에 대하여 알아보자. 위에서 언급하였듯이 가장 기본적인 대입연산자는 “=”가 맞으나 편리함을 위하여 사용하는 대입 연산자가 있다. 조금이라도 프로그래밍 언어의 길이를 줄이기 위한 편리하고, 컴퓨터 입장에서도 설계도가 짧아져 읽는 시간이 짧게 걸리므로 양측에게 유용한 연산자이다.

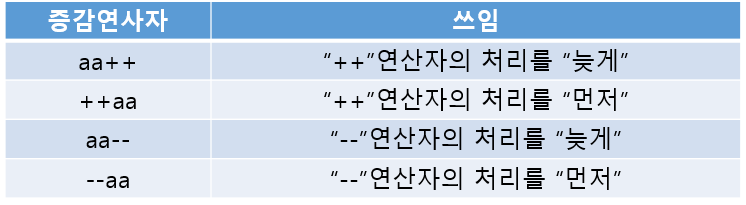


보다 쉬운 암기를 위한 팁이 있다면, 컴퓨터는 항상 위에서 아래로, 왼쪽에서 오른쪽 순서로 컴파일을 진행함을 생각하자. 컴퓨터의 입장에서 대입 연산자 뒤의 모든 것은 수식으로 인식하기 때문에 “=+”의 경우 + 연산자를 그저 피 연산자의 부호로 생각하므로 복합 대입 연산자에서 =은 항상 오른쪽에 위치한다.

(그림: 컴파일의 진행 경로를 그림으로 표현, =+의 경우와 +=의 경우를 둘 다 보여줌)

3-2-4. 증감 연산자

앞으로 산술연산자를 사용할 때 느끼겠지만 변수의 값을 1씩 추가하거나 1씩 감소시켜야 하는 프로그래밍 언어를 짜는 경우가 굉장히 많다. 그러할 때 항상 a=a+1을 표기함은 비경제적이므로, 위의 복합 대입 연산자와 같이 프로그래머와 컴퓨터 양측의 편리를 위하여 그러한 산술 연산자를 몇 개를 만들었다. 이 친구들을 증감 연산자라고 부르기도 한다.



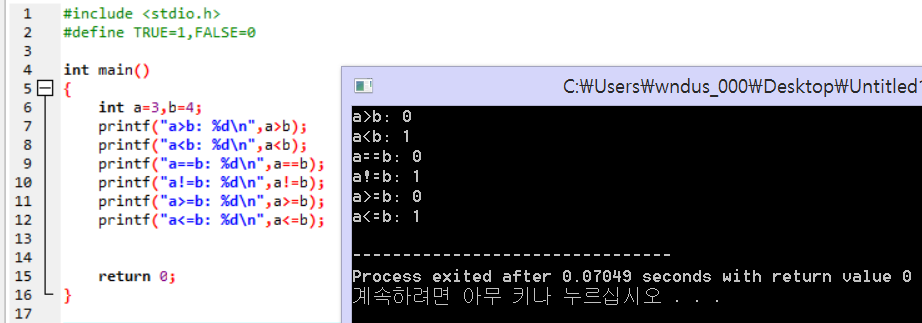
표를 보면 ++와 —를 변수의 앞, 뒤 어디에 붙이던 아무 문제가 없어 보이지만 사실 그렇지 않다. “++”를 변수의 “앞”에 붙인다는 의미는 그 해당 “연산”에서 그 어떠한 연산보다 “++”연산자의 처리를 “먼저”하겠다는 의미이다. 심지어 대입 연산자보다도 먼저! 같은 맥락으로 “++”를 변수의 “뒤”에 붙인다는 의미는 그 해당 “수식”에서 그 어떠한 연산보다 “++”연산자의 처리를 “늦게”하겠다는 의미이다. 대입 연산자보다도 늦게! 하지만 해당 수식이 끝나기 직전에 변수는 1이 증가한다. 그러하면 옆의 예시로 나온 결과값이 10, 12, 10, 8임을 쉽게 알 수 있다.

(그림: 소스코드를 통해 ++a와 a++ 차이 보여주기)

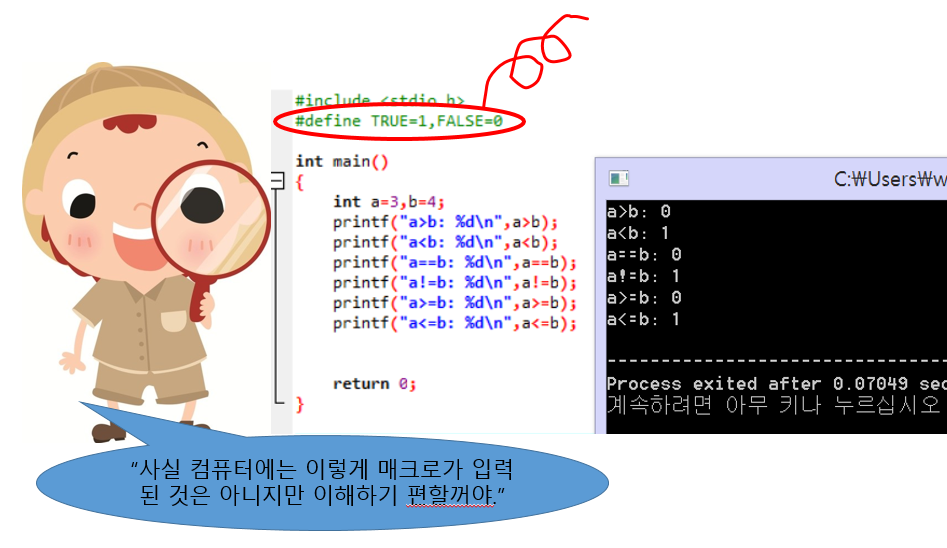
3-2-5. 비교 연산자

비교 연산자는 말 그대로 비교를 통한 결과가 참인지, 거짓인지 여부를 결과로 도출되는 연산자이다. c언어의 여섯 비교 연산자는 다음 표와 같다.



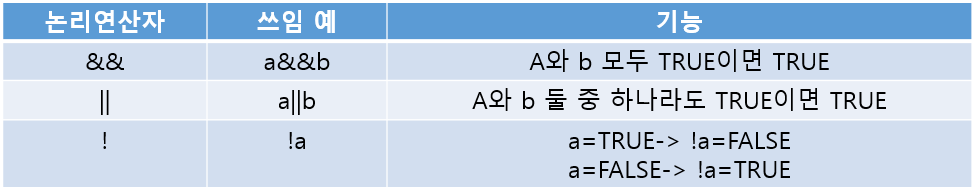


예시를 통해 알 수 있듯이, 연산자의 결과가 참인 경우에는 1이, 결과가 거짓인 경우에는 0이 출력된다. 앞의 변수단원에서 배운 매크로상수의 개념으로 이해하면 편리하다. 아래의 그림과 같이 TRUE=1, FALSE=0으로 지정했다고 생각하자.

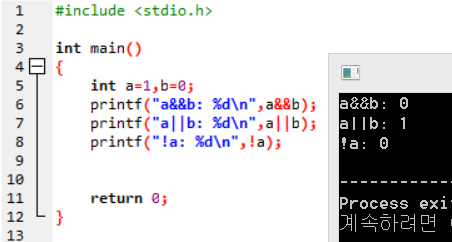


3-2-6. 논리 연산자

논리 연산자 또한 비교 연산자와 마찬가지로 논리의 결과가 참인지, 거짓인지를 결과로 도출하는 연산자이다. c언어의 세가지 논리연산자는 다음과 같다.



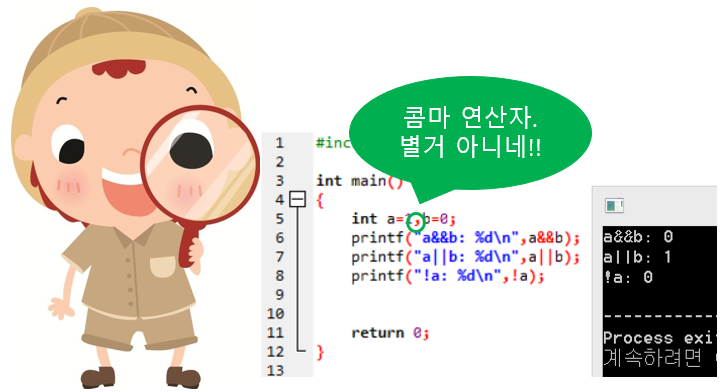
예시를 통해 알 수 있듯이, 논리 연산자도 비교 연산자의 결과처럼 참인 경우에는 1이, 결과가 거짓인 경우에는 0이 출력된다.



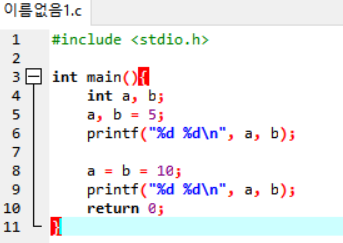
**:**

3-2-7. 기타 연산자

그 외의 연산자로는 형 변환 연산자, 콤마 연산자, sizeof 연산자, 조건 연산자 등이 있다. 우선 형 변환은 앞의 변수 단원에서 다루었는데, 자료형을 명시적으로 바꿔주는 것 역시 연산자로 보아 형 변환 연산자라고 부른다. 또 콤마 연산자는 여태까지 문제를 풀어왔다면, 프로그래밍을 하면서 수 많은 콤마”,”를 사용했을 것이다. 이 콤마 또한 연산자인데 이유는 앞과 뒤의 피 연산자를 구분해 주기 때문에 이항 연산자에 해당한다. C언어에서 사용되는 모든 콤마는 사실상 연산자인데, 이 개념이 잘 와 닿지 않을 수도 있지만 적응하도록 노력하자.



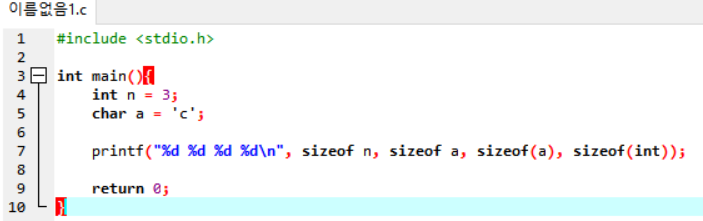
위의 그림과 같은 콤마 연산자이지만, 프로그래밍 시 주의해야 할 점이 하나 있다.



이 예제 소스를 실행시켜보라.

첫 줄의 경우 a, b는 다른 값이 나오게 되고, 둘째 줄의 경우 a, b 모두 같은 값이 나오게 된다. 간혹 몇몇 초보 프로그래머들이 a, b = 5; 라고하면 a와 b 모두에 5가 대입될 것이라고 착각하는 경우가 있는데, 이는 아직 독자 여러분들이 C언어에서의 연산자 우선순위에 대해 학습하지 못해서 발생하는 결과이다. a와 b 모두에 같은 값을 동시에 대입하려면 아래 두 번째 대입문과 같이 a = b = 특정값; 꼭 이러한 형태로 해주어야만 원하는 결과값을 얻을 수 있으므로 주의하자. 연산자 우선순위는 이번 단원의 마지막 장에서 다루니 꼭 정독하여 자기 것으로 만들기 바란다.

그럼 sizeof 연산자와, 조건 연산자가 남았다. 그 중 먼저 sizeof 연산자에 대해 알아보자면, 이는 이름에서도 알 수 있듯 변수 혹은 자료형의 크기를 구하는 연산자이다.



위의 예제 소스 코드를 통해 살펴보면, 변수의 크기를 구하는 경우

sizeof 변수명 또는 sizeof(변수명)

이러한 형태가 모두 되지만, 자료형의 크기를 직접 sizeof 연산자로 구하는 경우에는 무조건 괄호를 씌워 주어야만 컴파일 에러가 발생하지 않는다.

sizeof(자료형)

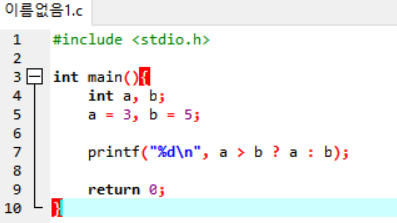
그럼 변수 혹은 자료형의 크기를 구하는 sizeof 연산자에 대해서도 간략히 알아보았고, 기타 연산자 소단원에서는 마지막으로 조건 연산자에 대해 알아보려고 한다.

지금까지 배운 모든 연산자 중 증감 연산자와 논리 연산자 중 !(논리NOT) 연산의 경우는 항이 하나여서 단항, 그리고 이외의 모든 연산자들은 항이 두개인 이항 연산자였다. 그렇지만 C언어에서는 유일한 삼항 연산자가 있는데, 이것이 바로 조건 연산자이다. 사용 방법은 다음과 같다.

조건식A ? 조건식A가참인경우실행 : 조건식A가거짓인경우실행 ;

이런 형태를 띤다.

예제 소스를 통해 조금 더 살펴보도록 하자.



이 예제는 a, b 중 더 큰 수를 출력하는 예제인데, 조건식A에는 비교 연산자를 이용한 조건식이 들어가고, a가 더 작으므로 콜론 뒤의 문장이 실행되어 b=5가 출력이 된다.

@심화: 비트 연산자(~, |, ^, &, >>, <<)

비트 연산자란 이름에서 알 수 있듯이 비트 단위로 값을 조작하는 연산자이다. 앞의 변수 단원에서 변수의 메모리에 대하여 바이트, 비트 설명을 참고하고 오면 공부하기 편할 것 이다.

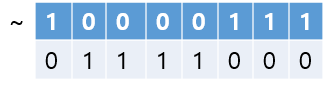
이 연산자가 필요한 이유는 기계어나 어셈블리 언어의 입장에서 가능한 기능을 c 프로그램에서도 가능하게 하기 위해서이다. 생각을 해보면, 컴퓨터가 정말 사람의 계산법인 +와 –를 직접 연산하는 것은 아닐 것이다. 그렇다면 c언어는 필요하지 않으니깐...! 그들은 계산 시에 인간의 논리인 덧셈과 뺄셈을 직접 실행하는 것이 아니고, 변수에 저장된 데이터의 비트를 조작하여 결과값을 출력한다. 즉 아래의 예제처럼, +1을 처리하기 위하여 컴퓨터는 해당 명령을 인식한 뒤에, 비트를 바꾸는 것이다. +연산을 실행하는 것 보다는 비트 연산자를 통하여 같은 기능을 발현하는 것이 훨씬 컴퓨터가 구동하기 편리하고 처리 속도가 빠르다.

(그림: 컴퓨터가 덧셈을 위해 고민한 뒤 연산하는 모습과 비트 연산자를 간단히 알아듣고 실행하는 모습)

이러한 비트연산자에는 여섯 가지 종류가 있다. 하나하나 예시를 통하여 살펴보자.

**~연산자 (비트NOT 연산자)**

단항 연산자로, 하나의 피 연산자를 받아 모든 비트에 들어있는 숫자를 보수로 바꿔준다. 즉 1->0, 0->1로 보수를 취하여 바꾸는 연산자이다.



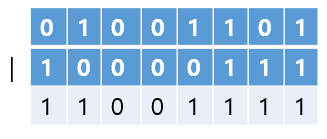
(그림: “~” 1의 보수 비트연산자, 실제 실행화면을 보임.)

이 연산자의 경우 논리NOT 연산자(!)와 혼동하지 말 것을 당부한다. 논리 NOT의 경우 0이 아닌 모든 수에 대해 !를 취하면 0으로, 0에 대해 !를 취하면 1로 바꾸는 간단한 연산자인 반면 비트 NOT의 경우는 모든 비트를 바꿔주는 연산자이므로 보통 음수가 나오는 경우가 많다.

**|연산자 (비트OR 연산자)**

이항 연산자로, 두 개의 피 연산자를 받아 OR 연산을 수행한다. 1을 참으로, 0을 거짓으로 생각하면, OR연산자의 결과가 매 비트마다 출력된다.

(그림: OR 연산자의 결과 보여주기)

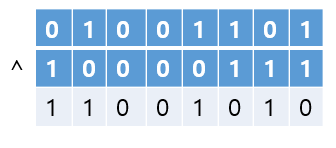


아래의 예제와 같이, 특정 비트를 1로 바꾸는데 유용하다.

(그림: “|” OR 비트연산자, 실제 실행화면을 보임.)

**^연산자 (비트XOR 연산자)**

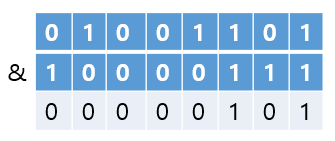
이항 연산자로, 두 개의 피 연산자를 받아 Exclusive OR 연산을 수행한다. Exclusive OR 연산은 두 수가 같으면 1, 다르면 0으로 계산한다. Exclusive OR 연산의 결과는 아래의 그림과 같다. 참고하자.



(그림: “^” XOR 비트연산자, 실제 실행화면을 보임.)

**&연산자 (비트AND 연산자)**

이항 연산자로, 두 개의 피 연산자를 받아 & 연산을 수행한다. 1을 참으로, 0을 거짓으로 생각하면, &연산자의 결과가 매 비트마다 출력된다.



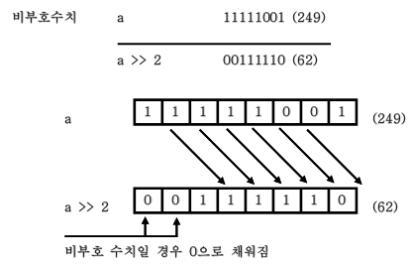
(그림: “&” AND 비트연산자, 비트마다 하나의 상자로 표현, 두 줄을 비교해 바뀌는 경로를 보임.)

|예제와는 정 반대로, 아래의 예제처럼 특정 비트를 0으로 바꾸는데 유용하다,

(그림: “&” AND 비트연산자, 실제 실행화면을 보임.)

**>>연산자 (비트이동 연산자1)**

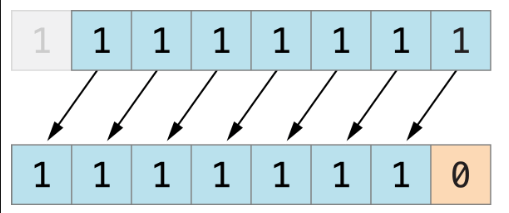
단항 연산자이다. 원하는 만큼 피 연산자의 비트를 오른쪽으로 이동시킨다. 예제를 통해 이해하자.



(그림: “>>” 왼쪽 시프트 연산자, 실제 실행화면을 보임.)

**<<연산자 (비트이동 연산자2)**

단항 연산자이다. 원하는 만큼 피 연산자의 비트를 왼쪽으로 이동시킨다. 예제를 통해 이해하자.

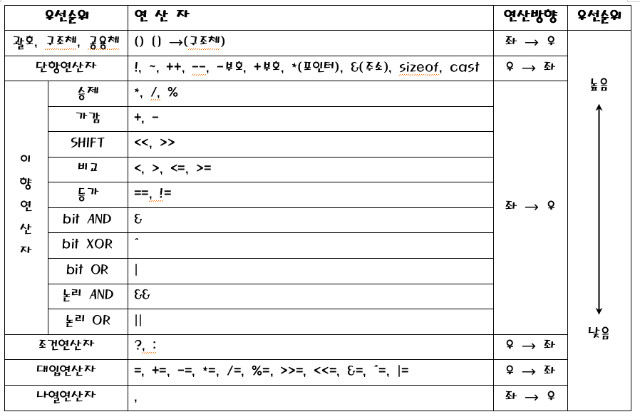


(그림: “<<” 오른쪽 시프트 연산자, 실제 실행 화면을 보임.)

참고로 한가지 더 알려주자면, 비트 이동 연산자의 경우도 앞에서 다루었던 복합 대입 연사자처럼 <<=, >>= 를 이용할 수 있다. 평소에 잘 사용하는 것은 아니니 지금은 알아 두기만 하자.

3-3. 연산자 우선순위

주어지는 사칙연산을 한번 수행해보자. 4 + 8 \* 2 = ? 답이 20임은 모든 사람이 쉽게 알 수 있을 것이다. 왜냐하면 곱셈이 덧셈보다 먼저 연산이 되어야 하는 것이 바로 사칙연산의 규칙이기 때문이다. 이와 같이 c언어의 연산자에도 우선순위가 있다.



가구를 만드는 과정으로 생각해보면, 목재와 같은 재료를 가공할 때에는 분명 순서에 따라 가공해야 한다. 먼저 톱으로 큰 틀을 자르고 사포로 면을 다듬어야 순서에 맞지 사포로 면을 다듬고 톱으로 자른다면 이상한 결과가 나오는 것이다. 그와 같이 연산자의 변수 가공에도 순서가 정해져 있다.

이제 3단원이 마무리를 지었고 C언어의 약 20%를 학습하였다. 이번 장까지는 C언어를 이용하는 프로그래밍이 어떤 것인지, 또 C에서 이용하는 변수와 그 변수들을 다루는 연산자들에 대해 알아보았다. 앞으로 몇 단원은 지금까지 배운 것을 바탕으로 프로그램의 논리적인 흐름을 조작하는 방법에 대해 배울 것이다. 그럼 이제 1, 2, 3단원을 통해 배운 것들을 다시 한번 복습하고, 연관된 문제들을 풀어보며 단단하게 학습을 한 뒤 4장으로 넘어가도록 하자!