C언어 연산자

대입 연산자와 산술 연산자

종류	기호	문법	의미
대입 연산자	=	a=3	a에 3을 대입
산술 연산자	+	a+3	a와 3의 합
	-	a-3	a와 3의 차
	*	a*3	a와 3의 곱
	/	a/3	a를 3으로 나눈 몫
	%	a%3	a를 3으로 나눈 나머지

비교 연산자, 논리 연산자의 쓰임

종류	기호	문법	의미
비교 연산자	==	a==3	a와 3이 같은가?
	!=	a!=3	a와 3이 다른가?
	>, <	a>3, a<3	a가 3보다 큰(작은)가?
	>=, <=	a>=3, a<=3	a가 3보다 크거나(작거)나 같은가?
논리 연산자	&&	a&&b	a와 b가 모두 참일 경우에만 참
	||	a||b	a와 b가 모두 거짓일 경우에만 거짓
	!	!a	a가 참이면 거짓, 거짓이면 참

비트연산자

비트 연산자를 사용할 경우 프로그램의 용량을 매우 많이 줄일수 있으며 속도가 증가되는 장점을 가지고 있음

```
#include <stdio.h>
main ()
{
    printf ( "%d %d \n", 3 & 1, 3 | 1);
    printf ( "%d \n", 3 ^ 1);
    printf ( "%d %d \n", 3 << 1, 3 >> 1);
}

결과값을 생각해보라
```

&

두 값 비트 자릿수를 비교해, 두 값 모두에 1이 있을때만 1을, 나머지 0

|

두 값중 하나가 1이면 1을, 둘다 0 이면 0

^

두 자리 값이 값으면 0, 다르면 1

>> <<

쉬프트로 비트를 쉬프트함

~

비트열의 값들을 각각 부정(not)을 하라는 뜻으로 각 자리수의 비트값을 반대로 바꾸어 1의 보수로 변환시킴

보수

보수는 보충해주는 수 (반대라고 생각하면 됨)

그럼 1의보수와 2의보수가 있는데 그 차이점은?

- 1의 보수 : 그냥 0->1 ... 1->0 으로 바꾸면됨
- 2의 보수 : 1의보수한결과를 +1를 해주면 됨

왜 2의 보수를 사용할까 ?

- 음수를 '2의 보수 표현' 방식으로 저장하거나 출력한다.
- 9를 32비트로 만들어 저장시키면
- 0000 0000 0000 0000 0000 0000 0000 1001
- 이걸 1의 보수 취하면
- 1111 1111 1111 1111 1111 1111 1111 0110
- 이걸 2의 보수 취하면
- 1111 1111 1111 1111 1111 1111 1111 0111

-9를 위와 같이 1111 1111 1111 1111 1111 1111 1111 0111 으로 표현함

근데 이부분에서 1의보수를 볼때 2의보수를 볼때 값이 달라지는건 사실

~ 연산자를 사용하면 2의보수로 변환되어 저장된다는 뜻임

삼항 연산자

Java에서도 있는거

(조건문) ? 참 : 거짓;

엄청 유용해서 익숙하면 디게 편함

복합 대입 연산자와 증감 연산자

종류	기호	문법	의미
복합 대입 연산자	+=	a+=1	a=a+1
	-=	a-=1	a=a-1
	*=	a*=1	a=a*1
	/=	a/=1	a=a/1
	%=	a%=1	a=a%1
	&=	a&=1	a=a&1
	^=	a^=1	a=a^1
	|=	a|=1	a=a|1
	<<=,>>=	a <<=,>>=1	a=a <<=,>>= 1
증가 연산자	++	a++ or ++a	a=a+1
감소 연산자	--	--a or a--	a=a-1

참조 연산자

참조 연산자는 변수가 사용하는 메모리의 주소값을 나태는 연산자

즉 포인터 ...

```
#include <stdio.h>
main ()
{
    int a = 3;
    printf ( "%d \n", &a);
    printf ( "%d %d \n", a, *&a);
}
```

%a : 변수 a가 저장된 메모리(RAM) 공간의 주소값

*%a : a의 메모리 주소에 지정된 값을 나타내는 연산자

(*a 만사용하게 되면 오류가 발생한다)

포인터 (살짝)

포인터 변수에는 메모리 주소가 저장되어 있음.

이때 메모리 주소가 있는 곳으로 이동해서 값을 가져오고 싶다면 "*"를 사용하여 가져와야함

형변환 연산자

형 변환 연산자는 자료형을 바꾸어 주기 위해 사용되는 연산자

(int) (float) (double) 이런거

우선순위

기능별 분류	연산자	우선 순위
일차식	(), [], .	1
단항 연산자	!, ~, ++, −, (자료형)	2
승제 연산자	*, /, %	3
가감 연산자	+, −	4
시프트 연산자	<<, >>	5
비교 연산자	<, <=,>,> =	6
등가 연산자	==, !=	7
비트 연산자	& (8), ^ (9), | (10)	8, 9, 10
논리 연산자	&& (11), || (12)	11, 12

if문과 switch의 차이점

- if문은 모든 비교연산(조건설정)이 가능하지만 switch문은 값에 의해서만 다른 처리가 가능하다
- if문은 블록{}으로 영역을 구분하지만 switch문은 case와 break로 영역을 구분
- if문은 else문으로 switch문은 default문으로 그 외 나머지 경우를 처리함
- if문은 모든 변수형을 사용 가능하지만, switch문은 실수형을 사용할 수 없다.

---

do ~ while

일단 do 블록을 한번 실행한 후 반복 조건을 확인한다.

while이랑 do~while의 차이점은 무엇인지 한번 생각해보기 ~!