3주차 1차시 비트 연산자

【학습목표】

1. 비트 연산자에 대해 설명할 수 있다.
2. C언어 제공 기본 자료형에 대해 설명할 수 있다.

학습내용1 : 비트 연산자

1. 비트연산자(비트 이동 연산자)

연산자	연산자의 기능	결합방향
&	비트단위로 AND 연산을 한다. 예) num1 & num2;	→
|	비트단위로 OR 연산을 한다. 예) num1 | num2;	→
^	비트단위로 XOR 연산을 한다. 예) num1 ^ num2;	→
~	단항 연산자로서 피연산자의 모든 비트를 반전시킨다. 예) ~num; // num은 변화 없음, 반전 결과만 반환	←
<<	피연산자의 비트 열을 왼쪽으로 이동시킨다. 예) num<<2; // num은 변화 없음, 두 칸 왼쪽 이동 결과만 반환	→
>>	피연산자의 비트 열을 오른쪽으로 이동시킨다. 예) num>>2; // num은 변화 없음, 두 칸 오른쪽 이동 결과만 반환	→

2. & 연산자: 비트단위 AND

```
int main(void)
{
    int num1 = 15;     // 00000000 00000000 00000000 00001111
    int num2 = 20;     // 00000000 00000000 00000000 00010100
    int num3 = num1 & num2;     // num1과 num2의 비트단위 & 연산
    printf("AND 연산의 결과: %d \n", num3);
    return 0;
}
```

- 0 & 0 0을 반환
- 0 & 1 0을 반환
- 1 & 0 0을 반환
- 1 & 1 1을 반환

```
AND 연산의 결과: 4
```

```
      00000000 00000000 00000000 00001111
& 연산 00000000 00000000 00000000 00010100
-------------------------------------------
      00000000 00000000 00000000 00000100
```

3. | 연산자: 비트단위 OR

```
int main(void)
{
    int num1 = 15;    // 00000000 00000000 00000000 00001111
    int num2 = 20;    // 00000000 00000000 00000000 00010100
    int num3 = num1 | num2;
    printf("OR 연산의 결과: %d \n", num3);
    return 0;
}
```

- 0 & 0 0을 반환
- 0 & 1 1을 반환
- 1 & 0 1을 반환
- 1 & 1 1을 반환

```
OR 연산의 결과: 31
```

```
      00000000 00000000 00000000 00001111
| 연산 00000000 00000000 00000000 00010100
-------------------------------------------
      00000000 00000000 00000000 00011111
```

4. ^ 연산자: 비트단위 XOR

```
int main(void)
{
    int num1 = 15;    // 00000000 00000000 00000000 00001111
    int num2 = 20;    // 00000000 00000000 00000000 00010100
    int num3 = num1 ^ num2;
    printf("XOR 연산의 결과: %d \n", num3);
    return 0;
}
```

- 0 & 0 0을 반환
- 0 & 1 1을 반환
- 1 & 0 1을 반환
- 1 & 1 0을 반환

```
XOR 연산의 결과: 27
```

```
        00000000 00000000 00000000 00001111
^ 연산  00000000 00000000 00000000 00010100
-------------------------------------------
        00000000 00000000 00000000 00011011
```

5. ~ 연산자

```
int main(void)
{
    int num1 = 15;    // 00000000 00000000 00000000 00001111
    int num2 = ~num1;
    printf("NOT 연산의 결과: %d \n", num2);
    return 0;
}
```

- ~ 0 1을 반환
- ~ 1 0을 반환

```
NOT 연산의 결과: -16
```

~ 연산 전 00000000 00000000 00000000 00001111

~ 연산 후 11111111 11111111 11111111 11110000

6. << 연산자: 비트의 왼쪽 이동(Shift)

num1 << num2　num1의 비트열을 num2칸씩 왼쪽으로 이동시킨 결과를 반환
8 << 2　정수 8의 비트열을 2칸씩 왼쪽으로 이동시킨 결과를 반환

```
int main(void)
{
    int num = 15;     // 00000000 00000000 00000000 00001111

    int result1 = num<<1;     // num의 비트 열을 왼쪽으로 1칸씩 이동
    int result2 = num<<2;     // num의 비트 열을 왼쪽으로 2칸씩 이동
    int result3 = num<<3;     // num의 비트 열을 왼쪽으로 3칸씩 이동

    printf("1칸 이동 결과: %d \n", result1);
    printf("2칸 이동 결과: %d \n", result2);
    printf("3칸 이동 결과: %d \n", result3);
    return 0;
}
```

```
1칸 이동 결과: 30
2칸 이동 결과: 60
3칸 이동 결과: 120
```

```
00000000 00000000 00000000 00011110    // 10진수로 30
00000000 00000000 00000000 00111100    // 10진수로 60
00000000 00000000 00000000 01111000    // 10진수로 120
```

왼쪽으로 한 칸씩 이동할 때마다 정수의 값은 두 배씩 증가한다.
반대로 오른쪽으로 한 칸씩 이동할 때마다 정수의 값은 반으로 줄어든다.

7. >> 연산자: 비트의 오른쪽 이동(Shift)

```
11111111 11111111 11111111 11110000      // -16
```

CPU에 따라서 달라지는 연산의 결과

```
00111111 11111111 11111111 11111100      // 0이 채워진 경우

11111111 11111111 11111111 11111100      // 1이 채워진 경우
```

왼쪽 비트가 0으로 채워진 음수의 경우 오른쪽으로 비트를 이동시킨 결과는 CPU에 따라서 달라진다. 따라서 호환성이 요구되는 경우에는 >> 연산자의 사용을 제한해야 한다.

```
int main(void)
{
    int num = -16;    // 11111111 11111111 11111111 11110000
    printf("2칸 오른쪽 이동의 결과: %d \n", num>>2);
    printf("3칸 오른쪽 이동의 결과: %d \n", num>>3);
    return 0;
}
```

```
2칸 오른쪽 이동의 결과: -4
3칸 오른쪽 이동의 결과: -2
```

학습내용2 : C언어 제공 기본 자료형

1. 자료형은 데이터를 표현하는 방법이다.

√실수를 저장할 것이냐? 정수를 저장할 것이냐!
• 값을 저장하는 방식이 실수냐 정수냐에 따라서 달라지기 때문에 용도를 결정해야 한다.

√얼마나 큰 수를 저장할 것이냐!
• 큰 수를 표현하기 위해서는 많은 수의 바이트가 필요하다
위의 두 사항은 이름 이외에 메모리 공간의 할당에 있어서 필요한 두가지 정보이다.

√아! 제가 정수를 저장할건데요. 크기는 4바이트로 하려고 합니다. 그 정도면 충분할 거에요.
그리고 변수의 이름은 num으로 할게요.."
제대로 된 요청이다.

위의 내용에 대한 C언어에서의 예는 int num; 이다.

√자료형의 수는 데이터 표현방법의 수를 뜻한다. C언어가 제공하는 기본 자료형의 수가 10개라면,
C언어가 제공하는 기본적인 데이터 표현방식의 수는 10개라는 뜻이 된다.

2. 기본 자료형의 종류와 데이터의 표현범위

자료형		크기	값의 표현범위
정수형	char	1바이트	−128이상 +127이하
	short	2바이트	−32,768이상 +32,767이하
	int	4바이트	−2,147,483,648이상 +2,147,483,647이하
	long	4바이트	−2,147,483,648이상 +2,147,483,647이하
	long long	8바이트	−9,223,372,036,854,775,808이상
			+9,223,372,036,854,775,807이하
실수형	float	4바이트	$\pm 3.4\times10^{-37}$이상 $\pm 3.4\times10^{+38}$이하
	double	8바이트	$\pm 1.7\times10^{-307}$이상 $\pm 1.7\times10^{+308}$이하
	long double	8바이트 이상	double 이상의 표현범위

√컴파일러에 따라서 약간의 차이를 보인다.
C의 표준에서는 자료형별 상대적 크기를 표준화 할 뿐 구체적인 크기까지 언급하지는 않는다.

√크게 정수형과 실수형으로 나뉘다.
데이터를 표현하는 방식이 정수형과 실수형 두 가지로 나뉘므로!

√정수형에도 실수형에도 둘 이상의 기본 자료형이 존재한다.
표현하고자 하는 값의 크기에 따라서 적절히 선택할 수 있도록 다수의 자료형이 제공!

3. 연산자 sizeof를 이용한 바이트 크기의 확인

```
int main(void)
{
    int num = 10;
    int sz1 = sizeof(num);
    int sz2 = sizeof(int);
    . . . .
}
```

√변수 num과 int의 크기를 계산하여 그 결과로 sz1과 sz2를 초기화하고,
sizeof 연산자의 피연산자로는 변수, 상수 및 자료형의 이름 등이 올 수 있다. 소괄호는 int와 같은 자료형의 이름에만 필수! 하지만 모든 피연산자를 대상으로 소괄호를 감싸주는 것이 일반적!

```c
int main(void)
{
    char ch=9;
    int inum=1052;
    double dnum=3.1415;
    printf("변수 ch의 크기: %d \n", sizeof(ch));
    printf("변수 inum의 크기: %d \n", sizeof(inum));
    printf("변수 dnum의 크기: %d \n", sizeof(dnum));

    printf("char의 크기: %d \n", sizeof(char));
    printf("int의 크기: %d \n", sizeof(int));
    printf("long의 크기: %d \n", sizeof(long));
    printf("long long의 크기: %d \n", sizeof(long long));
    printf("float의 크기: %d \n", sizeof(float));
    printf("double의 크기: %d \n", sizeof(double));
    return 0;
}
```

```
변수 ch의 크기: 1
변수 inum의 크기: 4
변수 dnum의 크기: 8
char의 크기: 1
int의 크기: 4
long의 크기: 4
long long의 크기: 8
float의 크기: 4
double의 크기: 8
```

4. 정수의 표현 및 처리를 위한 일반적 자료형 선택

√일반적인 선택은 int이다.
- CPU 가 연산하기에 가장 적합한 데이터의 크기가 int형의 크기로 결정된다. 연산이 동반이 되면 int형으로 형 변환이 되어서 연산이 진행된다. 따라서 연산을 동반하는 변수의 선언을 위해서는 int로 선언하는 것이 적합하다.

√char형 short형 변수는 불필요한가?
- 연산을 수반하지 않으면서(최소한의 연산만 요구가 되면서) 많은 수의 데이터를 정장해야 한다면 그리고 그 데이터의 크기가 char 또는 short로 충분히 표현이 가능하다면 char 또는 short로 데이터를 표현 및 저장하는 것이 적합하다.

```
int main(void)
{
    char num1=1, num2=2, result1=0;
    short num3=300, num4=400, result2=0;

    printf("size of num1 & num2: %d, %d \n", sizeof(num1), sizeof(num2));
    printf("size of num3 & num4: %d, %d \n", sizeof(num3), sizeof(num4));
    printf("size of char add: %d \n", sizeof(num1+num2));
    printf("size of short add: %d \n", sizeof(num3+num4));

    result1=num1+num2;
    result2=num3+num4;
    printf("size of result1 & result2: %d, %d \n", sizeof(result1), sizeof(result2));
    return 0;
}
```

```
size of num1 & num2: 1, 1
size of num3 & num4: 2, 2
size of char add: 4
size of short add: 4
size of result1 & result2: 1, 2
```

+ 연산결과의 크기가 4바이트인 이유는 피연산자가 4바이트 데이터로 형 변환 되었기 때문이다.

5. 실수의 표현 및 처리를 위한 일반적 자료형 선택

√실수 자료형의 선택기준은 정밀도이다. 실수의 표현범위는 float, double 둘 다 충분히 넓다.
그러나 8바이트 크기의 double이 float보다 더 정밀하게 실수를 표현한다.

√일반적인 선택은 double이다.
컴퓨팅 환경의 발전으로 double형 데이터의 표현 및 연산이 덜 부담스럽다. float형 데이터의 정밀도는 부족한 경우가 많다.

실수 자료형	소수점 이하 정밀도	바이트 수
float	6자리	4
double	15자리	8
long double	18자리	12

```
int main(void)
{
    double rad;
    double area;
    printf("원의 반지름 입력: ");
    scanf("%lf", &rad);
    area = rad*rad*3.1415;
    printf("원의 넓이: %f \n", area);
    return 0;
}
```

```
원의 반지름 입력: 2.4
원의 넓이: 18.095040
```

double형 변수의 출력 서식문자 %f printf
double형 변수의 입력 서식문자 %lf scanf

6. unsigned를 붙여서 0과 양의 정수만 표현

정수 자료형	크기	값의 표현범위
char	1바이트	-128이상 +127이하
unsigned char		0이상 (128 + 127)이하
short	2바이트	-32,768이상 +32,767이하
unsigned short		0이상 (32,768 + 32,767)이하
int	4바이트	-2,147,483,648이상 +2,147,483,647이하
unsigned int		0이상 (2,147,483,648 + 2,147,483,647)이하
long	4바이트	-2,147,483,648이상 +2,147,483,647이하
unsigned long		0이상 (2,147,483,648 + 2,147,483,647)이하
long long	8바이트	-9,223,372,036,854,775,808이상 +9,223,372,036,854,775,807이하
unsigned long long		0이상 (9,223,372,036,854,775,808 + 9,223,372,036,854,775,807)이하

√정수 자료형의 이름 앞에는 unsigned 선언을 붙일 수 있다. unsigned가 붙으면, MSB도 데이터의 크기를 표현하는데 사용이 된다. 따라서 표현하는 값의 범위가 양의 정수로 제한이 되며 양의 정수로 두 배 늘어난다.

【학습정리】

1. 비트 단위로 AND 연산을 하는 비트 연산자는 &

2. 비트 단위로 OR 연산을 하는 비트 연산자는 |

3. 비트 단위로 XOR 연산을 하는 비트 연산자는 ^

4. 단항 연산자로서 피연사자의 모든 비트를 반전시키는 비트 연산자는 ~

5. 피연산자의 비트 열을 왼쪽으로 이동시키는 비트 연산자는 <<

6. 피연산자의 비트 열을 오른쪽으로 이동시키는 비트 연산자는 >>