

# 데이터 표현 방식 상수와 기본 자료형

20200408

scanf??

# scanf의 사용법

```
#include <stdio.h>

int main(void)
{
    int num;

    scanf("%d", &num);

    printf("입력받은 수: %d\n", num);

    return 0;
}
```

# Practice 1

#scanf #입력 값 받기 #관계(비교) 연산자

숫자 2개를 입력 받아서  
산술 연산자 연산을 수행하시오.  
(+, -, \*, /, % 사용)

그리고 입력한 두 수가 같은 지  
여부도 확인하시오.  
(== 사용)

## 출력 예시

```
10 + 3 = 13  
10 - 3 = 7  
10 * 3 = 30  
10 / 3 = 3  
10 % 3 = 1
```

```
입력한 두 수가 같은가? (같으면 1 / 다르면 0) : 0
```

# Summary

1. scanf 뒤에 & (주소 연산자) 까먹으면 안돼 `scanf("%d", &num);`

# 진수와 바이트 개념



2 진수	8 진수	10 진수	16 진수
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E

# 8진수와 16진수를 이용한 데이터 표현

```
int num1 = 10; // 특별한 선언이 없으면 10진수 표현  
int num2 = 012; // 0 으로 시작하면 8진수로 인식  
int num3 = 0x43; // 0x 로 시작하면 16진수로 인식
```



# 비트와 바이트

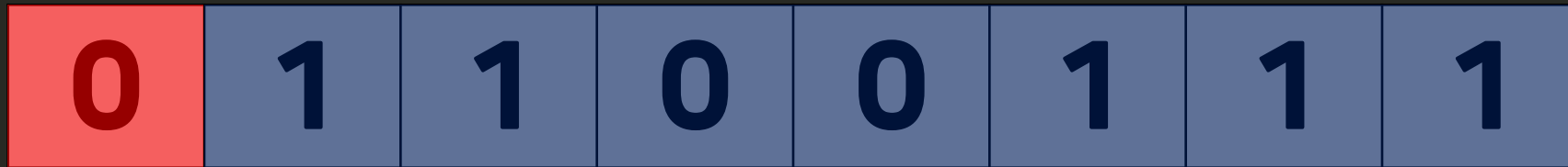
0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

1 Bit

1 Byte

0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

Most  
Significant  
Bit



부호  
비트

정수의 크기 표현

# 예제

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	1	1	1	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

1	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

# ★ 음수를 표현하는 방법 ★

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

5

1	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

-5

---

1	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---

?

# ★ 음의 정수 표현방법 = 2의 보수법 ★

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

Step 1



1의 보수를 취한다.

1	1	1	1	1	0	1	0
---	---	---	---	---	---	---	---

Step 2



1을 더한다.

1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---

# ★ 음의 정수 표현법 증명 ★

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

 5

1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---

 -5

---

1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

 0

# Practice 2

#음의 정수 표현방법 #잘못된 음수 표현 #2의 보수법

문제 1.

양의 정수 01001111과 00110011은 각각 10진수로 얼마인가?

★ 문제 2.

음의 정수 10101001과 11110000은 각각 10진수로 얼마인가?

# Summary

1. scanf 뒤에 & (주소 연산자) 까먹으면 안돼 `scanf("%d", &num);`
2. 음의 정수 표현 방법 (1의 보수 취하기 -> +1 해주기)



# 비트 연산자 (Bitwise Operator)

연산자	설명 및 예시	결합 방향
&	Bit 단위로 AND 연산	->
	Bit 단위로 OR 연산	->
^	Bit 단위로 XOR 연산	->
~	단항 연산자 모든 비트를 반전시키는 역할	<-
<<	비트 열을 왼쪽으로 이동	->
>>	비트 열을 오른쪽으로 이동	->

# & 연산자 : 비트단위 AND 연산

```
#include <stdio.h>

int main(void)
{
    int num1 = 15;           // 00000000 00000000 00000000 00001111
    int num2 = 20;           // 00000000 00000000 00000000 00010100

    int num3 = num1 & num2; // 00000000 00000000 00000000 00000100

    printf("AND 연산의 결과: %d\n", num3);

    return 0;
}
```

# | 연산자 : 비트단위 OR 연산

```
#include <stdio.h>

int main(void)
{
    int num1 = 15;           // 00000000 00000000 00000000 00001111
    int num2 = 20;           // 00000000 00000000 00000000 00010100

    int num3 = num1 | num2; // 00000000 00000000 00000000 00011111

    printf("OR 연산의 결과: %d\n", num3);

    return 0;
}
```

# ^ 연산자 : 비트단위 XOR 연산

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int num1 = 15;           // 00000000 00000000 00000000 00001111
```

```
    int num2 = 20;           // 00000000 00000000 00000000 00010100
```

```
    int num3 = num1 ^ num2;  // 00000000 00000000 00000000 00011011
```

```
    printf("XOR 연산의 결과: %d\n", num3);
```

```
    return 0;
```

```
}
```

# ★ ~ 연산자 : 비트단위 NOT 연산 ★

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int num1 = 15;           // 00000000 00000000 00000000 00001111
```

```
    int num2 = ~num1;        // 11111111 11111111 11111111 11110000
```

```
    printf("NOT 연산의 결과: %d\n", num2);
```

```
    return 0;
```

```
}
```

## 음의 정수 크기 확인하기

2진수로 표현된 음의 정수는 양의 정수와 달리 그 값의 크기를 바로 계산하기가 쉽지 않다. 따라서 이러한 경우에는 2의 보수를 취해서 크기를 확인해야 한다. 예를 들어서 다음과 같이 표현된 정수가 있다고 가정해보자.

11111111 11111111 11111111 11110000 // -16

이는 MSB가 1이니 음수이다. 그럼 이제 크기를 계산해 보자. 이를 위해서 2의 보수를 취하겠다. 그리고 그 결과는 다음과 같다.

00000000 00000000 00000000 00010000 // +16

이는 양의 정수이니 그 크기가 16임을 쉽게 확인할 수 있다. 따라서 11111111 11111111 11111111 11110000은 -16이라는 결론이 나온다. 참고로 2의 보수를 취하는 것은 -1을 곱하는 결과로 이어진다. 따라서 음의 정수에 2의 보수를 취하여 그 크기를 확인할 수 있는 것이다.

# << 연산자 : 비트의 왼쪽 이동

```
#include <stdio.h>

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

    int res1 = num1 << 1;        // 00000000 00000000 00000000 00011110
    int res2 = num1 << 2;        // 00000000 00000000 00000000 00111100
    int res3 = num1 << 3;        // 00000000 00000000 00000000 01111000

    printf("1칸 이동 결과: %d\n", res1);
    printf("2칸 이동 결과: %d\n", res2);
    printf("3칸 이동 결과: %d\n", res3);

    return 0;
}
```

# >> 연산자 : 비트의 오른쪽 이동

```
#include <stdio.h>

int main(void)
{
    int num1 = -16;           // 11111111 11111111 11111111 11110000

    int res1 = num1 >> 2;     // 11111111 11111111 11111111 11111100
    int res2 = num1 >> 3;     // 11111111 11111111 11111111 11111110

    printf("1칸 이동 결과: %d\n", res1);
    printf("2칸 이동 결과: %d\n", res2);

    return 0;
}
```



# ★ 논리 연산자 vs 비트 연산자 ★

```
#include <stdio.h>

int main(void)
{
    int num1 = 10;
    int num2 = 12;
    int res1, res2, res3;

    res1 = ((num1==10) && (num2==12));
    res2 = ((num1<12) || (num2>12));
    res3 = (!num1);

    printf("res1의 결과: %d\n", res1);
    printf("res2의 결과: %d\n", res2);
    printf("res3의 결과: %d\n", res3);

    return 0;
}
```

```
#include <stdio.h>

int main(void)
{
    int num1 = 15;           // 00000000 00000000 00000000 00001111
    int num2 = 20;           // 00000000 00000000 00000000 00010100

    int num3 = num1 & num2; // 00000000 00000000 00000000 00000100

    printf("AND 연산의 결과: %d\n", num3);

    return 0;
}
```

## 논리 연산자

(참 or 거짓) && (참 or 거짓)

-> 참, 거짓끼리의 연산!!

## 비트 연산자

00001111

00010100

-> 비트끼리의 연산!!

# Summary

1. scanf 뒤에 & (주소 연산자) 까먹으면 안돼 `scanf("%d", &num);`
2. 음의 정수 표현 방법 (1의 보수 취하기 -> +1 해주기)
3. ~ (NOT 연산) 주의하기
4. 논리 연산자 vs 비트 연산자

# 기본 자료형

# C언어의 기본 자료형

	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	$\pm 3.4 \times 10^{-37}$ 이상 $\pm 3.4 \times 10^{+38}$ 이하
	double	8바이트	$\pm 1.7 \times 10^{-307}$ 이상 $\pm 1.7 \times 10^{+308}$ 이하
	long double	8바이트 이상	double 이상의 표현범위

[표 05-1: C언어의 기본 자료형]

# Sizeof 연산자

```
#include <stdio.h>
```

```
int main(void)
{
```

```
    char ch = 'a';
    int num = 10;
```

```
    // 변수의 크기 확인
```

```
    printf("변수 ch의 크기: %d\n", sizeof(ch)); // 1
```

```
    printf("변수 num의 크기: %d\n", sizeof(num)); // 4
```

```
    // 자료형의 크기도 얼마인지 확인 가능
```

```
    printf("char의 크기: %d\n", sizeof(char)); // 1
```

```
    printf("int의 크기: %d\n", sizeof(int)); // 4
```

```
    return 0;
```

```
}
```

Sizeof( )는 함수가 아니라 연산자다!!

# ★ 정수를 처리하기 위한 일반적인 자료형 ★

```
#include <stdio.h>

int main(void)
{
    // char에 관한 변수 정의
    char num1 = 1, num2 = 2;
    char res_char = 0;

    // short에 관한 변수 정의
    short num3 = 300, num4 = 400;
    short res_short = 0;

    // 각 char 자료형의 크기
    printf("sizeof num1: %d\n", sizeof(num1));
    printf("sizeof num2: %d\n\n", sizeof(num2));

    // 각 short 자료형의 크기
    printf("sizeof num3: %d\n", sizeof(num3));
    printf("sizeof num4: %d\n\n", sizeof(num4));

    // 계산하는 순간??
    printf("sizeof char add: %d\n", sizeof(num1 + num2));
    printf("sizeof short add: %d\n\n", sizeof(num3 + num4));

    // res라는 변수에 값을 담은 후의 크기
    res_char = num1 + num2;
    res_short = num3 + num4;
    printf("sizeof res_char: %d\n", sizeof(res_char));
    printf("sizeof res_short: %d\n\n", sizeof(res_short));

    return 0;
}
```



왜 더하는 순간 sizeof( )는  
우리가 아는 결과대로  
나오지 않을까??



일반적으로 CPU가 처리하기에 가장 적합한 크기의  
정수 자료형을 int로 정의한다. 따라서 int형의 연산속도가  
다른 자료형의 연산속도에 비해 동일하거나 더 빠르다.



int형 연산이 CPU가 성능을 내기에 가장 좋은 연산!!

따라서 int보다 작은 크기의 데이터도

int형 데이터로 바뀌어서 연산이 진행된다!!

연산의 대상이 되는 변수를 선언하는 경우에는,  
특히 연산의 횟수가 빈번한 경우에는 저장되는 값의 크기가  
작더라도 int형 변수를 선언하는 것이 좋다!!

변수에 1이라는 값을 저장하고 싶은데

그렇다고 우리가 char 형식의 자료형을 쓰는 것은 아니잖아유?

char num = 1; <-> int num = 1;

그럼 char형 변수와 short형 변수는 왜 있는거야??

데이터의 양이 많아서 연산속도(int형 변수)보다  
데이터의 크기를 줄이는 것이 더 중요한 데이터들!!

정수 자료형에서는 int를 보편적으로 선택하듯이,  
실수 자료형에서도 보편적으로 선택하는 자료형은 없나유?

# 실수를 처리하기 위한 일반적인 자료형

실수 자료형	소수점 이하 정밀도	바이트 수
float	6자리      정밀도 너무 낮음	4
double	15자리	8
long double	18자리   크기 너무 커서 부담 多	12

[표 05-2: 실수 자료형의 정밀도]

# Practice 3

#실수를 대표하는 자료형 #윤희남 교수님 책 예제

다음과 같은 수식의 값을 계산하여 화면에 출력하는 프로그램을 작성 하시오. 지수 표기법을 사용하여 변수들을 초기화한다.

$$3.32 * 10^{-3} + 9.76 * 10^{-8}$$

출력 결과

0.003320

$3.32 * 10^{-3}$ 는  $3.32e-3$  으로 표기하면 된다.

# Summary

1. scanf 뒤에 & (주소 연산자) 까먹으면 안돼 `scanf("%d", &num);`
2. 음의 정수 표현 방법 (1의 보수 취하기 -> +1 해주기)
3. ~ (NOT 연산) 주의하기
4. 논리 연산자 vs 비트 연산자
5. 정수를 대표하는 자료형은 int (because of 연산속도)
6. 실수를 대표하는 자료형은 double (because of 정밀도)



# unsigned

- 정수 자료형의 이름 앞에만 unsigned를 붙일 수 있다.
- unsigned가 붙으면, 부호 비트(MSB)도 데이터의 크기를 표현하는데 사용이 된다.
- 따라서 표현할 수 있는 값이 0 이상의 범위로 두 배가 된다.

# ASCII Code

# 문자는 이렇게 표현되는 거구나!

```
int main(void)
{
    char ch1 = 'A';
    char ch2 = 'C';
}
```



```
int main(void)
{
    char ch1 = 65;
    char ch2 = 67;
}
```



# Practice 4 #아스키 코드 #%c의 사용법

다음 코드를 분석하시오.

하지만 일반적으로  
문자는 char에,  
정수는 int에 넣는다!

```
#include <stdio.h>

int main(void)
{
    char ch1 = 'A' // 문자를 char에 저장 (익숙)
    char ch2 = 65; // 정수를 char에 저장
    int ch3 = 'Z' // 문자를 int에 저장
    int ch4 = 90; // 정수를 int에 저장 (익숙)

    printf("%c %d\n", ch1, ch1);
    printf("%c %d\n", ch2, ch2);
    printf("%c %d\n", ch3, ch3);
    printf("%c %d\n", ch4, ch4);

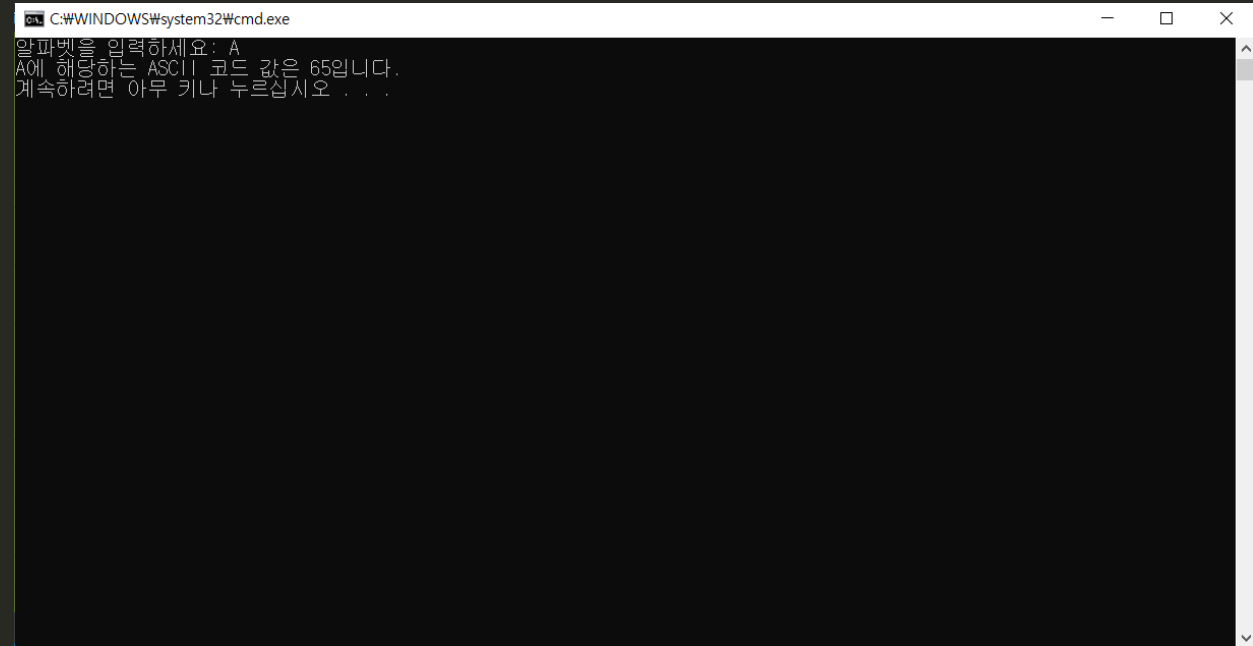
    return 0;
}
```

# Practice 5

#아스키 코드 #scanf #%c #문자 하나를 %c와 %d 둘 다 출력

프로그램 사용자로부터 알파벳 문자 하나를 입력 받는다. 그리고 이에 해당하는 아스키 코드 값을 출력하는 코드를 작성해보자.

예를 들어, 사용자가 문자 A를 입력하면 정수 65를 출력해야 한다.



```
C:\WINDOWS\system32\cmd.exe
알파벳을 입력하세요: A
A에 해당하는 ASCII 코드 값은 65입니다.
계속하려면 아무 키나 누르십시오...
```