3 일째

데이터타입 (int, short, char, float, double, long double)

int : 4 byte → 2^32 개 (약 42 억 9 천만) 표현 가능
shor t : 2 byte → 2^16 개 (65536)
char : 1 byte → 2^8 개 (256)
float : 4 byte
double : 8 byte
long double : 12 or 16 byte

ex) int data;
data = 3;
→ data 를 변수로 지정해도 메모리 공간을 차지 하지 않는다. 허나 data = 3 을 기입하는 순간, memory 공간을 잡고 3 을 저장한다.
(기억해두자: 2^8 = 256, 2^10 =1024, 2^16 = 65536, 4kb → 2^12)

* unsigned (x) → 음수값 존재 , unsigned (o) → 더 큰 값을 사용 가능

char =1byte, 2^8 = 256 → (-128 ~ 127 까지 표현)
unsigned char = 1bye, 2^8=256, → 양수 (0~ 255 까지 표현가능)

ex) char a;
a= 256; → a = -128 이 되버림. 최상위 비트가 부호비트로 1 이 되어 마이너스로 판단함.

* 음수 빨리만드는법 (2 의 보수 x)
숫자 1 과 -1 을 더하면 ? 0 이 나와야함. 음수로 만들 양수를 오른쪽에서 1 이 나올때까지 그대로 유지하고 처음 1 이 나온 1 을 그대로 떨궈주고 다음 수부터는 전부 바꿔준다.

```
 0000 0001   +1
 1111 1111   -1
-------------------
1 000 0000    0  ( 맨 앞의 1 은 버림 )

 0001 0100   +21
 1110 1100   -21
-------------------
1 0000 0000   0
```

Overflow & Underflow

오버 플로우 : 데이터 타입이 표현할 수 있는 최댓값의 범위가 위로 벗어날 경우, 맨 아래 비트로 내려가는 현상

언더 플로우 : 표현할 수 있는 최소값에서 아래로 내려갈 경우 반대로 맨 위로 올라가는 현상

ex) char 타입은 -128 ~ 127 이므로
127+1 = -128
-128 -1 = 127

ex1)

```c
#include <stdio.h>

int main(void)
{
char a = 127, b= -128;
a++;
b--;
printf("Overflow : %d\n", a);
printf("Underflow : %d\n", b);
return 0;
}
```

```
koitt@koitt-Z20NH-AS51B5U:~/my_proj$ gcc -g -O0 -o debug over.c
koitt@koitt-Z20NH-AS51B5U:~/my_proj$ ./debug
Overflow : -128
Underflow : 127
```

* 아스키코드표 는 왜 배우나?
- 인증 (암호화)를 하기위함. 문자가 숫자로 치환이 되기 때문에 암호화에 매우 효율적임.
기본적으로 우리가 일상에서 사용하는 모든 정보는 유선이 아니라 무선을 통해 많이 보급되고 있다.
문제는 SDR 이라는 근래 아주 핫한 무선 분야인데, 이것을 활용하면 무선상의 모든 데이터를 가로챌 수 있다.
암호화가 되어있지 않다면 ID, PWD 모두 털린다.

ex2) ‘A’ = 65, ‘B’ = 66, ‘C’ =67

```c
#include <stdio.h>

int main(void)
{
  int chr1 = 'A', chr2 = 'B', chr3 = 'C';
  printf("A의 해당하는 수는 = %d\n", chr1);
  printf("소문자는 = %c\n",chr1^32);
  printf("A-B+C = %c\n",chr1 - chr2 + chr3);
  return 0;
}
~
```

```
koitt@koitt-Z20NH-AS51B5U:~/my_proj$ vi exam1.c
koitt@koitt-Z20NH-AS51B5U:~/my_proj$ gcc -g -o0 -o debug exam1.c
koitt@koitt-Z20NH-AS51B5U:~/my_proj$ ./debug
A의 해당하는 수는 = 65
소문자는 = a
A-B+C = B
```

ex3)

```c
#include <stdio.h>

int main(void)
{
 int num1 = 10, num2 = 5;
 int result = num1 +num2;
 printf("result = %d\n",result);

 result = num1 - num2;
 printf("result = %d\n",result);

 result = num1 * num2;
 printf("result = %d\n",result);

 result = num1 % num2;
 printf("result = %d\n",result);

return 0;
}
```

```
result = 15
result = 5
result = 50
result = 0
```

ex4) printf 안에서 퍼센트를 출력하고 싶을 때,

```c
#include <stdio.h>

int main(void)
{
int num1 = 2, num2 = 2;
printf("num1%c=num2 = %d, num1= %d\n",37,num1%=num2,num1);
return 0;
}
```

→ 숫자 37 은 아스키코드로 ‘%’ 임.

```
num1%=num2 = 0, num1= 0
```

Bit operator

AND 서로 같으면 참
OR 하나만 참이면 참
XOR 서로 다르면 참
NOT 반대

예제 5)

```c
#include <stdio.h>

int main(void)
{
int num1 = 10;
printf("num1 << 1 = %d\n",num1 << 1);
printf("num1 << 3 = %d\n",num1 << 3);
printf("num1 >> 2 = %d\n",num1 >> 2);
printf("num1 & 3 = %d\n",num1 & 3);
printf("num1 | 16 = %d\n",num1 | 16);
printf("num1 ^ 5 = %d\n",num1 ^ 5);
printf("num1 ^ 3 = %d\n",num1 ^ 3);
printf("~num1 = %d\n", ~num1);
return 0;
}
```

결과

```
num1 << 1 = 20
num1 << 3 = 80
num1 >> 2 = 2
num1 & 3 = 2
num1 | 16 = 26
num1 ^ 5 = 15
num1 ^ 3 = 9
~num1 = -11
```

* 비트 연산자

AND, OR, XOR, NOT, Shift 연산

AND 는 서로 참으로 같을 때만 참,
OR 은 둘중 하나만 참이면 참
XOR 은 서로 달라야 참
NOT 은 반대
쉬프트 연산은 비트를 이동시키는 연산 (2 의승수) → 주의사항 : >> 는 나눌 때 소수점 자리를 그냥 버려버린다.

Ex6)

```c
#include <stdio.h>

int main(void)
{
  int num1 = 3, num2 = 7;

int result1,result2,result3;

result1=(num1 ==3 && num2 ==7);
result2=(num2 >100 || num2 <100);
result3 = !num2;

printf("result1= %d\n",result1);
printf("result2= %d\n",result2);
printf("result3= %d\n",result3);
return 0;
```

```
result1= 1
result2= 1
result3= 0
```

*숏컷의 이점

---

*sizeof()

1. sizeof(a)는 a 의

---

quiz

1. 입력을 6 을 주고 num <<4 를 수행하는 함수를 작성하고 결과를 출력과 같게 만든다.

```c
#include <stdio.h>
int one(int num)
{
int result;
   result=num<<4;
   printf("result=%d",result);
}

int main(void)
{   one(6);

   return 0;
}
```

→

```c
#include <stdio.h>
int one(int num)
{
   return num<<4;
}

int main(void)
{
   int num =6, res;
   res = one(num);
   printf("res = %d\n",res);

   return 0;
}
```

```
Q1. result= 96
```

2. num = 55 인경우 num>>3 은?

```c
#include <stdio.h>

int main(void)
{

   int num=55;
   int result= num>>3;
   printf("Q2. result= %d\n",result);

}
```

→

```c
#include <stdio.h>
int two(int num)
{
   return num>>3;
}

int main(void)
{
   int num =55, res;
   res = two(num);
   printf("res = %d\n",res);

   return 0;
}
```

```
Q2. result= 6
```

3. 1byte 타입 num1 = 21, num2 = 31 의 and, or, xor 결과

```c
#include <stdio.h>

int main(void)
{
char num1 = 21, num2 =31;
int result1= num1 & num2;
int result2= num1 | num2;
int result3= num1 ^ num2;
printf("Q3. result1=%d\n, result2=%d\n, result3=%d\n",result1,result2,result3);
return 0;
}
```

```
Q3. result1=21
, result2=31
, result3=10
```

→ 바꾼것은 다했는데 지워져서 생략 Q1,2 번과 마찬가지로 함수 3 개만들어서 return 사용하면됨

* scanf() function 안쓸꺼지만 임시방편으로 배운다.

키보드로 무언가를 입력하고자 할때 사용
printf 와 쌍을 이루는 녀석이다. scanf 의 첫번째 입력은 “%d” 혹은 “%f”, “%lf” 등이 올 수 있다.
그리고 두번째 입력은 반드시 결과를 받을 주소값을 적어야한다. 그렇기 때문에 주소값을 의미하는 &가 들어간것.
ex1)

```c
#include <stdio.h>

int main(void)
{
int num1;
float num2;
printf("정수를 입력해:");
scanf("%d",&num1);

printf("실수를 입력해:");
scanf("%f",&num2);


printf("입력된 2개의 data: %d, %f\n",num1,num2);

return 0;

}
```

```
정수를 입력해:3
실수를 입력해:4.3
입력된 2개의 data: 3, 4.300000
```

ex2)

```c
#include <stdio.h>

int main(void)
{
int num1,num2;
double real1,real2;
printf("2개 정수를 입력해:");
scanf("%d   %d",&num1,&num2);

printf("2실수를 입력해:");
scanf("%lf  %lf",&real1,&real2);


printf("입력된 4개의 data: %d, %d, %lf, %lf\n",num1,num2,real1,real2);

return 0;

}
```

```
2개 정수를 입력해:2 4
2실수를 입력해:3.5 4.3
입력된 4개의 data: 2, 4, 3.500000, 4.300000
```

* 조건문 if

ex1)

```c
#include <stdio.h>

int main(void)
{
int num;

printf("정수를 입력하세요:");
scanf("%d",&num);

if(num <0)
{
printf("입력된 수는 0미만\n");
}

if(num >= 0)
{
printf("입력된 수는 0 이상\n");
}
return 0;
}
```

```
정수를 입력하세요:2
입력된 수는 0 이상
koitt@koitt-Z20NH-AS51B5U:~/my_proj$ ./debug
정수를 입력하세요:-01
입력된 수는 0미만
```

ex2)

```c
#include <stdio.h>

int main(void)
{
int num;

printf("정수를 입력하세요:");
scanf("%d",&num);

if(num % 2)
printf("입력된 수는 홀수\n");


else
 printf("입력된 수는 짝수\n");
return 0;
}
```

```
정수를 입력하세요:3
입력된 수는 홀수
koitt@koitt-Z20NH-AS51B5U:~/my_p
정수를 입력하세요:4
입력된 수는 짝수
```

ex3)

```c
#include <stdio.h>

int main(void)
{
int num;

printf("정수를 입력하세요:");
scanf("%d",&num);

if(num == 0)
printf("입력된 수는 %d\n",num);


else if(num>0)
 printf("입력된 수는 양수 %d\n",num);

else
 printf("입력된 수는 음수 %d\n",num);


return 0;
}
```

```
정수를 입력하세요:-4
입력된 수는 음수 -4
koitt@koitt-Z20NH-AS51B5U:~/my_proj$ ./debug
정수를 입력하세요:2.5
입력된 수는 양수 2
koitt@koitt-Z20NH-AS51B5U:~/my_proj$ ./debug
정수를 입력하세요:0
입력된 수는 0
```

*switch 문

if 문으로 참과 거짓의 지옥을 만들게 되면 가독성이 심하게 떨어지는데 switch 는 case 부분에 조건이 있어서 case 에 해당하는 조건만 보고 해당 부분만 확인하면 되므로 if 문을 이용한 참과 거짓의 지옥보다는 가독성이 올라간다.(그래서 컴파일러를 switch 문을 써서 만든다.) 추가적으로 case 안에서의 동작은 break 를 만나기 전까지는 계속된다.

ex1)

```c
#include <stdio.h>
int main(void)
{
int num;

printf("상담은 1, 사용내역조회는 2, 계약 해지는 3을 눌러주세연");
scanf("%d",&num);

switch(num)
{
    case 1:
    printf("상담 센터에 연결중입니다.\n");break;
    case 2:
    printf("사용 내역을 조회합니다.\n");break;
    case 3:
    printf("계약 해지를 진행합니다.\n");break;
    default:
    printf("번호를 잘못 누르셨습니다.\n");break;
}
return 0;
}
```

```
상담은 1, 사용내역조회는 2, 계약 해지는 3을 눌러주세연2
사용 내역을 조회합니다.
```

*while 문은 무엇인가. 무한루프는 while 문 안에 계속 참이되는 값을 넣으면 무한루프를 돈다.
if 나 switch 와 같이 조건을 활용한다. 단 작업을 반복시킬 수 있고, 조건이 거짓이 될 때 까지 반복하게 된다.
(조건이 만족하는 동안 반복하게된다.)

ex1)

```c
#include <stdio.h>
int main(void)
{

int i=0, result = 'A';

while(i<10)
{
printf("%c\n",result);
result++;
i++;
}
return 0;
}
```

```
A
B
C
D
E
F
G
H
I
J
```

ex2) I 가 짝수 일때 if 문을 참으로 만들어 짝수 일때만 출력하게 만듬.

```c
#include <stdio.h>

void print_even(int start, int end)
{
        int i = start;

        while(i<=end)
        {
                if(!(i%2)){
                        printf("even = %d\n",i);
                }
                i++;
        }

        int main(void)
        {
                print_even(1,100)
        }
```

```
even = 56
even = 58
even = 60
even = 62
even = 64
even = 66
even = 68
even = 70
even = 72
even = 74
even = 76
even = 78
even = 80
even = 82
even = 84
even = 86
even = 88
even = 90
even = 92
even = 94
even = 96
even = 98
even = 100
```

http://cafe.naver.com/hestit/55

(반드시 함수를 만들어서 동작하게 만든다.)

2 번, 6 번, 8 번, 9 번, 11 번 빼 고,

12 번은 리눅스에서 디버깅 하는 방법을 정리한다. Gdb 상에서 아직 소개하지 않은 명령어들 bt, c 에 대해 조사해보고 활용해보자.