1. 배운내용 복습(goto, 파이프라인, for)
- goto
주로 Kernel에서 많이 사용하며
Buffer를 사용하는 곳에서 주로 사용함

밑예제와 같이 에러가 났을때 if문과 goto문의 대처차이

```
#include<stdio.h>
int main(void)
{           int i, j, k;
          for(i=0; i<5; i++)
          {
                    for(j=0; j<5; j++)
                    {
                              for(k=0; k<5; k++)
                              {
                                        if((i==2)&&(j==2)&&(k==2))
                                        {
                                                  printf("Error!!!₩n");
                                        }
                                        else
                                        {
                                                  printf("Data₩n");
                                        }
                              }
                    }
          }

}
```

if문의 경우

```
#include<stdio.h>
int main(void)
{           int i, j, k, flag = 0;
          for(i=0; i<5; i++)
          {
                    for(j=0; j<5; j++)
                    {
                              for(k=0; k<5; k++)
                              {
                                        if((i==2)&&(j==2)&&(k==2))
```

```
                                        {
                                                printf("Error!!!₩n");
                                                flag = 1; //작업을 종료해달라는 플래그
                                        }
                                        else
                                        {
                                                printf("Data₩n");
                                        }
                                        if(flag)
                                        {
                                                break;
                                        }
                                }
                                if(flag)
                                {
                                        break;
                                }
                        }
                        if(flag)
                        {
                                break;
                        }
                }
        }
```

goto문의 경우

```
#include<stdio.h>
int main(void)
{       int i, j, k;
        for(i=0; i<5; i++)
        {
                for(j=0; j<5; j++)
                {
                        for(k=0; k<5; k++)
                        {
                                if((i==2)&&(j==2)&&(k==2))
                                {
                                        printf("Error!!!₩n");
                                        goto err_handler;
                                }
                                else
```

```
                                        {
                                                printf("Data₩n");
                                        }
                                }
                        }
                }
        return 0;
err_handler:
        printf("Goto Zzang!₩n");
        return -1;
}
```

이와같이 if문과 다르게 goto문은 간결하게 에러에 대처가 가능하다
if문은 기본적으로 mov, cmp, jmp로 구성된다.
goto는 jmp 하나로 끝이다.

for문이 여러개 생기면 if, break 조합의 경우
for문의 갯수만큼 mov, cmp, jmp를 해야한다.
문제는 바로 jmp명령어다.
call이나 jmp를 CPU Instruction(명령어) 레벨에서
분기 명령어라고 하고 이들은 CPU 파이프라인에 매우 치명적인 손실을 가져다준다.

- 파이프라인
기본적으로 단순한 CPU의 파이프라인은 아래와 같은 3단계로 구성된다.
1. Fetch - 실행해야할 명령어를 물어옴
2. Decode - 어떤 명령어인지 해석함
3. Excute - 실제 명령어를 실행시킴

파이프라인은 짧게는 5단계부터 길게는 수십단계로 구성된다.
(ARM, Intel 등등 다양한 프로세서들 모두 마찬가지)

그런데 왜 jmp나 call 등의 분기 명령어가 문제가 될까?

jmp나 call등의 분기명령어가 문제가 되는이유로는
기본적으로 분기 명령어는 파이프라인을 부순다.

이뜻은 가장 단순한 CPU가 실행까지 3 clock을 소요하는데
파이프라인이 깨지니 또 다시 3 clock을 버려야함을 의미한다.
만약 파이프라인의 단계가 수십단계라면 분기가 여러번 발생하여
파이프라인 단계 x 분기 횟수만큼 CPU clock을 낭비하게된다.
즉 성능면에서 goto가 월등히 압도적이다.(jmp 1번)

- for
for문의 탄생배경은?
while문은 초기화, 조건식, 증감식이 보기 불편함
간결성의 미학
for(초기화; 조건식; 증감식)으로 구성됨
for 하나 써두면 이 세가지가다포함되있다.

```
#include<stdio.h>
int main(void)
{
int i, result;
for(i=0; result='A';  i<10; i++, result++)
{
printf("%c₩n",result);
}
return 0;
}
```

//c는 무조건 선언을 위에 다해놔야함
리눅스는 for(int i=0; ->xxx안됨

```
#include<stdio.h>
int main(void)
{
int i=0, result = 'A';
for(i=0;//초기화  i<10;//조건부  i++, result++//증감식)
{
printf("%c₩n",result);
}
return 0;
}
```

for문의 무한루프
조건이없으면 영원히 돔

```
#include<stdio.h>
int main(void)
{int i, result='A';
for(;;)
{
printf("%c₩n",result);
}
```

return 0;
}
//무한루프 끄는법 ctrl+c

2. 문제은행(어제 내용을 for문으로 작성)

-1~1000사이에 3의배수의 합을 구하시오

```c
#include <stdio.h>

// synthesis: first - start, second - end, three - times
int syn(int start, int end, int times)
{
        int res = 0, i = start;

        for(i=start; i<end; i++)
        {
                if(!(i % 3))
                {
                        res += i;
                }
                i++;
        }

        return res;
}

int main(void)
{
        printf("tot series sum = %d\n", syn(1, 1000, 3));
        return 0;
}
```

- 1 ~ 1000사이에 4나 6으로 나눠도 나머지가 1인 수의 합을 출력하라.

```c
#include <stdio.h>

int syn(int start, int end, int t1, int t2)
{
        int res = 0, i = start;

        for(i=start; i<end; i++)
                {
                if(((i % 4) == 1) || ((i % 6) == 1))
                {
                        res += i;
                }
                i++;
        }

        return res;
}

int main(void)
{
        printf("tot series sum = %d\n", syn(1, 1000, 4, 6));
        return 0;
}
```

- 구구단

```c
#include <stdio.h>

void print_rom(void)
{
        int i=2, j=1;

        for(i=start; i<end; i++)
        {
                for(j<10)
                {
                        printf("%d x %d = %d\n",i,j,i *j);
                        j++;

                }
                j=1;
                i++;
        }

}
int main(void)
{
        print_rom();
        return 0;
}
```

3. fib함수 동작분석(디버깅 및 그림 그리기)