오늘은 백준 브론즈 4 문제
https://www.acmicpc.net/problem/11282

11282번: 한글
문제 한글의 각 글자는 초성, 중성, 종성으로 이두어져 있고, 이 세 가지를 …
www.acmicpc.net

생각보다 어려운 문제였습니다
N을 입력하면 N번째 글자를 출력해야 하는 프로그램이기 때문에
현대 한글에서 표현할 수 있는 모든 글자 11172개가 담긴 배열이 필요하다고 생각해
처음에는 이중 포인터 배열을 만들어 한글을 일일이 담아야 하나 싶었습니다

다행히 하단 블로그에서
한글 유니코드를 UTF-8 인코딩으로 변환하는 방법을 알 수 있었고
글자 하나에 해당하는 문자 세 개를 번호 순서대로 숫자로 출력해본 결과
특정한 규칙을 발견할 수 있었습니다

https://yoonbh2714.blogspot.com/2015/06/cc-utf-8_19.html

C/C++ 한글 utf-8 출력
관심가는 것들을 기록하고 공유하는 소프트맨 블로그입니다.
yoonbh2714.blogspot.com

첫번째 글자 '가'에 해당하는 UTF-8 3바이트 문자열을 숫자로 나타내면 {-22, -80, -128}
마지막 11172번째 글자 힣에 해당하는 문자열은 {-19, -98, -93}입니다
문자열에서 1, 2 인덱스 숫자의 범위는 -128~-65이며
다음 번째 글자로 넘어갈 때마다 2 인덱스의 숫자는 1씩 증가하다가
-64가 되어 범위를 벗어나면 -128이 되고 그 앞의 인덱스인 1 인덱스 숫자가 1 증가합니다
1 인덱스도 마찬가지로 -64가 되면 -128이 되고 0 인덱스 숫자가 1 증가합니다

이러한 규칙을 반복문 형태로 나타낸 것이 아래 코드입니다
상단 블로그에 있는 코드는 시프트 연산자를 이용한 코드라서 이해하기 어려워
이해하기 쉽게 수정해보았습니다

```c
// C99
#include <stdio.h>

int main()
{
    int N; // 1 ≤ N ≤ 11,172
    char hangul[3] = {-22, -80, -128}; // 한글 유니코드(UCS) 0xAC00(가) ~ 0xD7A3(힣) 1117

    scanf("%d", &N);
    for (int i = 1; i < N; i++)
    {
        hangul[2]++;
        if (hangul[2] == -64)
        {
            hangul[2] = -128;
            hangul[1]++;
            if (hangul[1] == -64)
            {
                hangul[1] = -128;
                hangul[0]++;
            }
        }
    }
    printf("%c%c%c", hangul[0], hangul[1], hangul[2]);

    return 0;
}
```

그런데 해당 코드는 11172번째 글자를 출력하기 위해 반복문을 11171번이나 돌아야 합니다
물론 크게 의미없는 정도의 시간복잡도이긴 하지만
뭔가 아쉬워 몫/나머지 연산자를 이용해 간단하게 만들어보았습니다
이게 뭐라고 한 시간이 넘게 걸렸습니다

```c
#include <stdio.h>

int main()
{
    int N; // 1 ≤ N ≤ 11,172
    char hangul[3]; // 한글 유니코드(UCS) 0xAC00(가) ~ 0xD7A3(힣) 11172자를 UTF-8 인코딩으

    scanf("%d", &N);
    hangul[0] = (N + 3071) / 4096 - 22;
    hangul[1] = ((N + 3071) / 64) % 64 - 128;
    hangul[2] = (N - 1) % 64 - 128;
    printf("%c%c%c\n", hangul[0], hangul[1], hangul[2]);

    return 0;
}
```