[No. 6] **The Golden Key** ____년 __월 __일

문제

열 문제로 쪽지 시험을 보았다. 학생들의 답안지는 다음과 같이 주어졌다.

```
\anssheet{cheolsu}{1=1, 2=3, 3=4, 4=3, 5=5, 6=3, 7=1, 8=5, 9=4, 10=1}
\anssheet{yeonghi}{1=2, 2=2, 3=3, 4=5, 5=5, 6=3, 7=2, 8=2, 9=3, 10=2}
```

정답지는 다음과 같다.

```
1=2, 2=2, 3=3, 4=5, 5=3, 6=3, 7=1, 8=2, 9=3, 10=2
```

두 학생의 답안지를 처리하는 명령 `\anssheet`와 채점하는 명령 `\Score`를 작성하여라.

입력: `\Score{Cheolsu}`
출력: Cheolsu: 20

1 prop 자료형, property list

<*key*> = <*value*> 형식의 리스트를 property list라고 한다. 예를 들어

```
color = blue,
width = 5cm,
name = trivname
```

이러한 구조를 가진 <*key*> = <*value*> 리스트에서 등호의 왼쪽 값을 key, 오른쪽 값을 value라고 한다. 이것을 prop 변수에 입력해보자.

```
\prop_set_from_keyval:Nn \l_tmpa_prop
{
    color = blue,
    width = 5cm,
    name = trivname
}
```

이것으로 무엇을 할 수 있는가? 기본적으로 할 수 있는 일은 다음과 같다.

- `\prop_map_function:NN`, `\prop_map_inline:Nn`. 주의할 점은 이 때 호출되는 함수는 두 개의 인자를 취하는 것이어야 한다는 것이다. 첫 번째 인자가 key이고 두 번째 인자가 value이다.
- `\prop_get:NnN`, `\prop_item:Nn`. prop에서 주어지는 key의 value를 가져오는 명령이다. 앞의 것은 가져온 값을 N에 넣고 뒤의 것은 입력스트림에 남긴다는 차이가 있다.
- `\prop_pop:NnN`, `\prop_put:Nnn`, `\prop_put_if_new:Nnn`. 이름 그대로의 동작을 한다. `\prop_put`은 해당 키에 이미 값이 있으면 덮어쓰고 `\prop_put_if_new:`는 해당 키가 없을 때만 새로 만들고 값을 넣는다.

이외에도 물론 다양한 interface 함수가 제공된다. 간단한 mapping만 연습해보고 간다. 참고로 prop에는 `\prop_use:`가 없다. prop의 item들을 "출력"하려면 mapping이 가장 확실한 방법이다.
mapping 함수는 두 개의 인자를 받는 형식이어야 한다. 현재 item의 key가 #1, value가 #2이다. inline 함수에서도 마찬가지이다.

```
\ExplSyntaxOn
\int_zero:N \l_tmpa_int
\prop_set_from_keyval:Nn \l_tmpa_prop
{
    color = blue,
    width = 5cm,
    name = trivname
}

\prop_map_inline:Nn \l_tmpa_prop
{
    \int_incr:N \l_tmpa_int
    \int_use:N \l_tmpa_int )~
    key:~\uline{#1},~value:~\uwave{#2}\par
}

\bigskip

\cs_new:Npn \print_item:nn #1 #2
{
    [#1] \fbox{#2} \newline
}

\prop_map_function:NN \l_tmpa_prop
   \print_item:nn
\ExplSyntaxOff
```

1) key: color, value: blue
2) key: width, value: 5cm
3) key: name, value: trivname

[color] blue
[width] 5cm
[name] trivname

주어진 문제로 돌아와서, 우리는 Cheolsu와 Yeonghi라는 이름을 가지는 prop를 정의하는 명령 `\anssheet`를 먼저 만들기로 하자.

```
\ExplSyntaxOn
\cs_new:Npn \build_nameprop:nn #1 #2
{
    \prop_if_exist:cF { l_ #1 _prop }
    {
        \prop_new:c { l_ #1 _prop }
    }

    \prop_set_from_keyval:cn { l_ #1 _prop }
    {
        #2
```

```
    }
}

\NewDocumentCommand \anssheet { m m }
{
    \build_nameprop:nn { #1 } { #2 }
}

\anssheet{cheolsu}{1=1, 2=3, 3=4, 4=3, 5=5, 6=3, 7=1, 8=5, 9=4, 10=1}

%%% === test ===
\prop_map_inline:Nn \l_cheolsu_prop
{
    $ #1 = #2 $ \quad
}
\ExplSyntaxOff
```

$1 = 1 \quad 2 = 3 \quad 3 = 4 \quad 4 = 3 \quad 5 = 5 \quad 6 = 3 \quad 7 = 1 \quad 8 = 5 \quad 9 = 4 \quad 10 = 1$

채점을 하려면 정답지를 만들어야 한다.

```
\anssheet{default}{1=2, 2=2, 3=3, 4=5,
    5=3, 6=3, 7=1, 8=2, 9=3, 10=2}
\ExplSyntaxOn
%%% === test ===
\prop_map_inline:Nn \l_default_prop
{
    $ #1 = #2 $ \quad
}
\ExplSyntaxOff
```

$1 = 2 \quad 2 = 2 \quad 3 = 3 \quad 4 = 5 \quad 5 = 3$
$6 = 3 \quad 7 = 1 \quad 8 = 2 \quad 9 = 3 \quad 10 = 2$

이 케이스에서 key들은 모두 숫자이다. 따라서 간단히 다음처럼 하면 점수를 알 수 있다.

```
\ExplSyntaxOn
\int_new:N \l_score_int

\NewDocumentCommand \Score { m }
{
    \prop_if_exist:cTF { l_ #1 _ prop }
    {
        \int_zero:N \l_score_int
        \score_fn:c { l_ #1 _ prop }

        #1:~\int_eval:n { 10 * \l_score_int }
    }
    {
```

```
        Something~went~wrong.
    }
}

\cs_new:Npn \score_fn:N #1
{
    \int_step_inline:nn { \prop_count:N \l_default_prop }
    {
        \prop_get:NnN #1 { ##1 } \l_tmpa_tl
        \prop_get:NnN \l_default_prop { ##1 } \l_tmpb_tl
        \str_if_eq:eeT { \l_tmpa_tl } { \l_tmpb_tl }
        {
            \int_incr:N \l_score_int
        }
    }
}

\cs_generate_variant:Nn \score_fn:N { c }
\ExplSyntaxOff

%%%% ==== test ====
\anssheet{default}{1=2, 2=2, 3=3, 4=5, 5=3, 6=3, 7=1, 8=2, 9=3, 10=2}
\anssheet{Cheolsu}{1=1, 2=3, 3=4, 4=3, 5=5, 6=3, 7=1, 8=5, 9=4, 10=1}
\anssheet{Yeonghi}{1=2, 2=2, 3=3, 4=5, 5=5, 6=3, 7=2, 8=2, 9=3, 10=2}

\Score{Cheolsu} \par
\Score{Yeonghi}
```

Cheolsu: 20

Yeonghi: 80

위의 정의에서 `\score_fn:c`를 써야 하는데, 이것을 바로 정의하지 않고 먼저 `\score_fn:N`을 정의한 다음 이것의 variant를 생성한 부분을 유심히 보아라. 기본형의 사용자 명령은 `N`과 `n`만으로 정의하고 그것의 인자확장형이 필요하면 해당 variant를 `\cs_generate_variant:Nn`으로 생성하는 것이 좋은 함수 정의 방법이다. 그리 하지 않는다면 모든 인자의 확장 문제가 전적으로 작성자에게 맡겨져 있어서 복잡한 low level 확장 명령을 활용하지 않을 수 없는데 이런 작업을 하다 보면 오류를 피할 수 없다.

연습문제

기본 1-1. 우리는 이미 소인수분해를 해 보았다. 24를 소인수분해하면

$$24 = 2 \times 2 \times 2 \times 3$$

인데, 이것을

$$24 = 2^3 \times 3$$

과 같이 동일한 소인수의 지수형태로 나타내고 싶다.
주어진 수를 소인수분해하여 그 결과를 지수형태로 나타내는 명령 `\xfactor`를 작성하여라.

입력: `\xfacror{24}`
출력: $2^3 \times 3$

문제

배경색상, 전경색상, 폰트크기, 세 가지 옵션에 따라 주어지는 단어를 식자하는 명령 `\WordColor`를 정의하여라.

입력: `\WordColor[bgcolor=blue,fgcolor=yellow,fsize=Large]{expl3}`
출력: expl3

2 keys 자료형

LaTeX Command에 대한 복습 LaTeX에서 "명령"을 만들 적에는

```
\newcommand\foo[3][yellow]{%
    \colorbox{#1}{\color{#2}#3}}
\foo{black}{expl3} \foo[blue]{white}{expl3}
```

expl3 expl3

`\newcommand`의 지정인자 가운데 처음 한 개를 optional로 만들 수 있었다.
상황을 극적으로 바꾸어놓은 것은 xparse이다. 옵션 인자를 부여하는 것이 너무나 간단해졌고, 요즘 유행하는 옵션 인자를 뒤로 보내는 스타일의 명령을 정의하는 것도 어렵지 않게 되었다.

```
\NewDocumentCommand\ffoo{moo}{%
  \IfNoValueTF{#2}{\def\bccolor{yellow}}{\def\bccolor{#2}}%
  \IfNoValueTF{#3}{\def\fccolor{black}}{\def\fccolor{#3}}%
  \colorbox{\bccolor}{\color{\fccolor}#1}}
\ffoo{xparse} \ffoo{xparse}[red] \ffoo{xparse}[blue][white]
```

xparse xparse xparse

참고로, 예전에는 옵션 인자를 뒤로 보내는 것은 일종의 금기였다. 그 이유는

```
\mycommand{arg1}[normal]
```

이라고 썼을 때 이 [normal]이 해당 명령의 인자인지 아니면 그냥 텍스트인지 구분할 수 없다는 이유에서였다. 아다시피 예전 TeX 소스는 스페이스를 거의 남기지 않는 코딩 컨벤션이 있었고 그 때문에 조금이라도 오해의 여지를 남기지 않아야 할 필요가 있었기 때문이다.
한편 graphics의 keyval 패키지와 더불어 예컨대

```
\includegraphics[width=1cm,height=3cm]{demo}
```

와 같은 <*key*> = <*value*> 형식의 옵션 인자를 부여할 수 있게 되었으나 이를 코딩하는 것이 아주 편하지만은 않았다. 여기서 expl3의 keys 자료형을 배우게 된다.

기본 개념 우리가 하고 싶은 것은, 예를 들어

```
bgcolor=white, fgcolor=blue
```

라고 하였을 때, `\l_bgcolor_tl`이 `white`라는 값을 갖도록 하고 `\l_fgcolor_tl`의 값이 `blue`가 되도록 하자는 것이다. 그리고 명령을 정의하는 곳에서 이 두 `tl`을 쓰면 된다.

지금 정의하려는 <*key*> = <*value*> 그룹에 유니크한 이름을 부여하여야 한다. (이것을 module이라고 한다.) 그것을 `mytest`라고 부르기로 하고,

```
\keys_define:nn { mytest }
{
    bgcolor  .tl_set:N = \l_bgcolor_tl,
    fgcolor  .tl_set:N = \l_fgcolor_tl
}
```

정의가 이루어진 후에 여기에 실제로 값을 할당하기 위해서 다음 명령을 쓴다.

```
\keys_set:nn { mytest }
{
    bgcolor = white,
    fgcolor = blue
}
```

무슨 일이 일어났는지 다음 테스트 코드로 확인하자.

```
\ExplSyntaxOn
\keys_define:nn { mytest }
{
    bgcolor  .tl_set:N = \l_bgcolor_tl,
    fgcolor  .tl_set:N = \l_fgcolor_tl
}

\keys_set:nn { mytest }
{
    bgcolor = white,
    fgcolor = blue
}

\tl_use:N \l_bgcolor_tl \\
\tl_use:N \l_fgcolor_tl

\ExplSyntaxOff
```

white
blue

`\keys_define:nn`에서 쓸 수 있는 value 할당 명령에 주의하여야 한다. 우리가

```
\tl_set:Nn \l_bgcolor_tl { white }
```

이렇게 쓰는 것을 여기서는

```
bgcolor  .tl_set:N = \l_bgcolor_tl
```

로 쓰고 있으므로, 당연히

```
.int_set:N
.dim_set:N
.fp_set:N
```

과 같은 명령이 있을 것을 알 수 있다. 참고로 <*key*> = <*value*> 할당을 위하여 지정된 변수가 정의되고 있지 않다면 새로이 정의하는 일까지 하기 때문에 `\tl_new:N \l_bgcolor_tl` 같은 명령을 미리 둘 필요가 없다.
나아가

```
.clist_set:N
.prop_put:N
.tl_set_x:N
```

이 있다. `.tl_set_x:N`은 `\tl_set:Nx`하라는 것이다. g가 붙으면 전역변수에 할당하는 것이 될 것임은 짐작가능하므로 따로 설명하지 않는다. 주의: `.seq_set:N`은 존재하지 않는다. 그 이유는 `\seq_set:Nn`이란 명령이 없기 때문이다. seq와 별도로 clist가 존재하는 이유 가운데 하나이다.
두어 가지 더 설명하여 둔다.

```
.code:n
```

원하는 LaTeX 코드를 입력 스트림에 남길 수 있다. value로 들어오는 것을 #1로 사용한다. 간단한 예를 들어 이해해보자면,

```
testkey    .code:n = { \dim_set:Nn \l_test_dim { #1 } }
```

이것은

```
testkey    .dim_set:N = \l_test_dim
```

이것과 완전히 동일한 의미이다. 단, `.code:n`에서 사용되는 변수는 미리 선언되어 있어야 한다. (keys를 쓸 때 value를 스크래치 변수에 할당하는 것은 나중에 매우 곤란한 문제를 야기한다. 따라서 변수 이름을 별도로 설정하여 주도록 하여라.)

```
unknown
```

이 key는 주로 에러 처리를 위하여 사용한다. 즉 `\keys_define:nn`에서 설정하지 않은 key가 들어왔을 때를 위한 것이다. 보통은 다음과 같이 하여 무시한다.

```
unknown      .code:n = {}
```

여기까지 배운 것을 바탕으로 주어진 문제를 풀 수 있다.

```
\ExplSyntaxOn
\tl_new:N \l_fontsize_tl
\keys_define:nn { wordcolor }
{
    bgcolor   .tl_set:N = \l_bgcolor_tl ,
    fgcolor   .tl_set:N = \l_fgcolor_tl ,
    fsize     .code:n = { \tl_set_eq:Nc \l_fontsize_tl { #1 } }
}

\keys_set:nn { wordcolor }
{
    bgcolor = white,
    fgcolor = black,
```

```
    fsize = normalsize
}

\NewDocumentCommand \WordColor { o m }
{
    \IfValueTF { #1 }
    {
        \keys_set:nn { wordcolor } { #1 }
    }
    {
        \keys_set:nn { wordcolor }
        {
            bgcolor = white,
            fgcolor = blue,
            fsize = normalsize
        }
    }

    \colorbox { \l_bgcolor_tl }
      { \color { \l_fgcolor_tl } \l_fontsize_tl #2 }
}

\ExplSyntaxOff

\WordColor[bgcolor=cyan!30,fgcolor=red!80,fsize=sffamily]{test}
\WordColor[fgcolor=green]{test}
\WordColor{test}
```

test test test

<*key*> = <*value*>에 default 설정을 하는 방법은 무엇일까? 이후 `\keys_set:nn`이 실행되기 전에 미리 `\keys_set:nn`을 한 번 실행해두는 것이 가장 좋다. 위의 예에서 처음 나오는 `\keys_set:nn`은 기본값을 설정해주는 역할을 한다. `.initial:n`이라는 함수가 별도로 마련되어 있지만 `\keys_set:nn`을 이용하는 편이 나중에 관리하기 편하다.
그런데 위의 예에서 보듯이 아예 아무런 옵션을 주지 않으면 색상과 폰트가 기본값으로 설정되지만 옵션 중 일부만 주면 (두 번째 `\WordColor` 명령) 주지 않은 key는 이전에 실행할 때의 값을 유지한다. (동일 scope, 즉 같은 중괄호 범위 내에서). 위의 정의에서 변수들이 모두 local로 정의되어 있기 때문에 scope를 달리 하면 초기화되지만 같은 scope 내에서 반복 사용할 때 이전 값을 이어받게 할 것인지 reset할 것인지는 명령 설계의 문제이다.

연습문제

2-0. `\WordColor` 명령을 주어질 때마다 기본값을 리셋하여 옵션에 열거되지 않은 key의 value가 항상 초기값을 갖도록 정의해 보아라.

keys 데이터타입에는 이밖에도 key를 하위 그룹으로 분할한다든가 하는 복잡한 조작을 가능하게 하고 있지

만 여기서는 더 다루지 않겠다. 다만 choices에 관한 예를 하나 들어두고 마치기로 한다. 둘 이상의 선택도 가능하게 하는 `.multichoices`의 예도 interface3 문서에 나와 있다.

```
\ExplSyntaxOn
\tl_new:N \l_gender_tl
\keys_define:nn { mytest }
{
    name    .tl_set:N = \l_name_tl,
    gender  .choice:,
        gender/M  .code:n = { \tl_set:Nn \l_gender_tl { male } },
        gender/F  .code:n = { \tl_set:Nn \l_gender_tl { female } },
    age     .int_set:N = \l_age_int
}
\ExplSyntaxOff
```

이제 gender라는 key는 M 또는 F라는 값만을 갖는다. 이를 처리하기 위한 `.code:n`에서 원하는 조작을 할 수 있다. 이 방법 말고 `.choices:nn`을 이용하는 것도 있으며 interface3 문서에 상세한 설명이 나온다. 간단히 테스트해보자.

```
\ExplSyntaxOn
\tl_new:N \l_gender_tl
\keys_define:nn { mynamecard }
{
    name    .tl_set:N = \l_name_tl,
    gender  .choice:,
        gender/M  .code:n = { \tl_set:Nn \l_gender_tl { male } },
        gender/F  .code:n = { \tl_set:Nn \l_gender_tl { female } },
    age     .int_set:N = \l_age_int
}

\NewDocumentCommand \NameCard { m }
{
    \keys_set:nn { mynamecard } { #1 }
    \namecard_print:NNN \l_gender_tl \l_name_tl \l_age_int
}

\cs_new:Npn \namecard_print:NNN #1 #2 #3
%% #1(gender), #2(name) = tl, #3(age) = int
{
    \efbox  %%% requires \usepackage{efbox}
    [ backgroundcolor=cyan!30, linewidth=2pt, linecolor=gray!50 ]
    {
        \str_case_e:nn { #1 }
        {
            { male }   { Mssr.~ }
            { female } { Mme.~ }
```

```
        }
        \tl_use:N #2 ~( \int_use:N #3 )
    }
}
\ExplSyntaxOff

\NameCard{name=Cheolsu,age=19,gender=M}
\NameCard{age=33,gender=F,name=Yeonghi}
```

Mssr. Cheolsu (19) Mme. Yeonghi (33)

연습문제

기본 2-1. 다음과 같은 명령을 작성해보아라.
입력: `\mymatrix[col=3,row=3]{a,b,c,d,e,f,g,h}`
출력:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 0 \end{pmatrix}$$

① 만일 주어지는 인자의 수가 cols × rows보다 부족하면 마지막을 0으로 채우고 넘치면 무시한다.
② 옵션 인자가 주어지지 않으면 2 × 2 정방행렬을 그린다.

참고. 이 명령을 작성하기 위하여 oblivoir 클래스를 쓰고 있다면 KTUG Private Repository로부터 ob-mathleading 패키지를 설치하고 이를 `\usepackage` 하여라.

```
$ tlmgr install ob-mathleading
```

연습문제

발전 2-2. 철수가 속해 있는 프로그래밍 동아리에서는 모일 때마다 "이진수 게임"을 한다. 게임 참여자가 각각 돌아가면서 "1" 또는 "0"의 수를 말하는 게임인데, 다음과 같은 순서로 진행한다. 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, ... 이것은 0, 1, 10, 11, 100, 101, 110, 111, ... 과 같이 진행되는 수를 한 자리씩만 말하게 한 것이다. 참여자가 7명이고 철수는 세 번째 자리에 앉아 있다. 철수가 10번째 순번까지 말해야 할 숫자를 나열하여라.

입력: `\prtnum{member=7,position=3,turn=10}`
출력: 1, 1, 1, 0, 1, 0, 1, 0, 0, 1

연습문제

실력 2-3. 앞서 우리가 만들어본 두 개의 Ti*k*Z 명령은 실은 동일한 명령이었다. 이를 일반화하여 다음과 같은 명령을 만들어 보아라.
인자로 `angle`, `forward`, `walk`을 줄 수 있다.

```
\DWalk{angle=15,forward=10,walk=20}
```

이것은 매 번 15도 회전하여 10만큼 전진하는 일을 20회 반복한다.
단, 다음 조건을 추가하여라.

① `angle=random`이라고 하면 회전각을 매회 난수적으로 얻는다.

② `forward=1+1`이라고 하면 처음 전진 길이를 1로 하고 매회 1씩 증가시켜간다.

③ `forward=random`이라고 하면 5mm에서 30mm 사이의 길이를 난수적으로 얻어서 전진한다. 길이의 범위를 제어할 수 있도록 만드는 것은 어려운 일이 아니겠으나 일단 정해진 범위가 있다고 보고 명령을 작성하기로 하자.

힌트. esgutil 새 버전에 `\esg_split_plussign:nNN`이라는 명령을 만들어 두었다. 이 함수는

```
\esg_split_plussign:nNN { 2 + 3 } \l_a_tl \l_b_tl
```

이라고 하면 #1의 내용에서 + 기호를 기준으로 앞의 것(2)을 `\l_a_tl`에, 뒤의 것(3)을 `\l_b_tl`에 넣어준다.