연습문제 풀이와 해설

esg002

2019/07/05

연습문제

기본 1. 문제에서 제시한 `\foobox`에서 각 글자에 아래 예시와 같이 색상을 입혀보아라. 각 글자 사이에 1pt의 간격을 둔다.

1. 입력: `\foobox{world}`
   출력: w o r l d

```
\ExplSyntaxOn

\NewDocumentCommand \foobox { m }
{
    \tl_map_inline:nn { #1 }
    {
        \color_box_each_char:n { ##1 }
        \hskip1pt
    }
}

\cs_new:Npn \color_box_each_char:n #1
{
    \fcolorbox{red}{red}{\color{yellow} #1}
}

\ExplSyntaxOff

\foobox{world}
```

w o r l d

주 1. `\tl_map_inline:nn`을 사용하여 인자로 주어지는 텍스트를 `\l_tmpa_tl`에 넣지 않고 바로 인라인 함수로 mapping한 보기입니다. `\l_tmpa_tl`에 넣고 `\tl_map_inline:Nn`하는 것도 당연히 인정. 그렇게 해야 할 때가 더 많습니다.

주 2. `\hskip1pt` 대신 `\hspace{1pt}`도 좋습니다.

주 3. `\fcolorbox` 대신 `\colorbox{red}{\color{yellow}#1}`도 인정. 이 둘의 차이는 frame을 색상을 주어 칠할 것인가입니다.

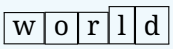
이 코드는 마지막 항목 다음에도 1pt가 붙습니다. 마지막 글자 다음에는 이 간격을 주고 싶지 않다면, 가장 쉬운 방법은 integer 하나를 사용하는 것입니다.

```
    \int_zero:N \l_tmpa_int
    \tl_map_inline:nn { #1 }
    {
        \color_box_each_char:n { ##1 }
        \int_incr:N \l_tmpa_int
        \int_compare:nT { \l_tmpa_int < \tl_count:n { #1 } }
          { \hskip1pt }
    }
```

[보충] 이재호 군의 해결책 1에 관하여 한편, 이재호 군이 보여준 여러 해법 중에서 다음과 같은 코드

```
\ExplSyntaxOn
\cs_new:Npn \foo_fn_rec:x #1
  {
    \tl_set:Nx \l_tmpa_tl { #1 }
    \tl_if_empty:oF { \l_tmpa_tl }
      {
        \fbox { \tl_head:N \l_tmpa_tl }
        \foo_fn_rec:x { \tl_tail:N \l_tmpa_tl }
      }
  }

\foo_fn_rec:x { world }
\ExplSyntaxOff
```



아이디어는 훌륭합니다. 매번 tl의 첫 글자를 "떼어내어서" 처리하고 버린다는 거지요. 약간의 코멘트를 붙여둡니다.

코멘트 1 인자형 지시자 :x가 그 의미를 가지려면 인자를 확장하는 코드가 함수 정의에 포함되어야 합니다. 이런 확장지시를 갖는 함수를 구성하는 방법에 대해서는 나중에 배우게 되는데, 여기서는 단순히 :n으로 정의하는 것이 좋겠다는 것만 지적합니다.

코멘트 2 `\tl_if_empty:oF`에서 어차피 tl이 변수로 되어 있으므로 `\tl_if_empty:NTF \l_tmpa_tl` 로 충분합니다.

이 아이디어를 조금 발전시켜 보겠습니다. 먼저, 입력받은 문자열의 head만 떼어내는 함수를 하나 정의합시다.

```
\ExplSyntaxOn
\cs_new:Npn \my_tl_shift:NN #1 #2
{
    \tl_set:Nx #2 { \tl_head:N #1 }
    \tl_set:Nx #1 { \tl_tail:N #1 }
}

%% === test ====
\tl_set:Nn \l_tmpa_tl { abc }
\my_tl_shift:NN \l_tmpa_tl \l_tmpb_tl
\tl_use:N \l_tmpb_tl,~ \l_tmpa_tl \\
\my_tl_shift:NN \l_tmpa_tl \l_tmpb_tl
```

```
\l_tmpb_tl,~ \l_tmpa_tl
\ExplSyntaxOff
```

a, bc
b, c

`\my_tl_shift:NN`은 첫 번째 인자로 들어오는 tl의 첫 글자를 두 번째 인자 tl에 넣고 원래의 tl에서 첫 글자를 제거합니다.

참고: 사실 이런 역할을 하는 함수가 clist와 seq에는 이미 있어요. tl은 간단히 작성할 수 있기 때문에 없다고 불평할 것도 없습니다.

이 함수가 있으므로 다음처럼 할 수 있게 되었습니다.

```
\ExplSyntaxOn
\NewDocumentCommand \fooboxa { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \int_do_until:nn { \tl_count:N \l_tmpa_tl <= 0 }
    {
        \my_tl_shift:NN \l_tmpa_tl \l_tmpb_tl
        \fbox { \l_tmpb_tl }
    }
}
\ExplSyntaxOff

\fooboxa{world}
```

w o r l d

문제를 해결하는 아이디어는 이재호 군의 원래 발상과 동일합니다. 그런데 이쪽이 expl3스럽다고 생각할 수 있어요. 이번 주 과제인 `\int_do_until`을 여기서 썼습니다.
어쨌든, expl3로 코딩하는 한, 재귀적 호출과 반복 실행은 항상 그 "종료조건이 한 눈에 들어오도록" 작성하지 않으면 안 됩니다.

연습문제

기본 2. `\foobox`에서 인자로 주어진 글자 수를 세어서 마지막에 괄호와 함께 표현하는 명령 `\barbox`를 작성하여라.

발전 3. 기본 문제 2번의 색상상자를 홀수번째 오는 문자에만 적용하도록 `\baroddbox` 명령을 작성하여라. 문자 사이에는 1pt의 간격을 둔다.

---

2. 입력: `\barbox{world}`
   출력: w o r l d (5)
3. 입력: `\oddbarbox{world}`
   출력: w o r l d (5)

2번은 생략하고 3번만 해결해봅니다. 글자를 칠하는 명령은 앞서 정의한 것을 쓰겠습니다.

```
\ExplSyntaxOn

\NewDocumentCommand \baroddbox { m }
{
    \int_zero:N \l_tmpa_int

    \tl_map_inline:nn { #1 }
    {
        \int_incr:N \l_tmpa_int
        \int_if_odd:nTF { \l_tmpa_int }
        {
            \color_box_each_char:n { ##1 }
        }
        {
            ##1
        }
        \hspace{1pt}
    }

    {}~( \tl_count:n { #1 } )
}

\ExplSyntaxOff
\baroddbox{world}
```

---

w o r l d (5)

여기서 어려운 점은 없는 걸로 생각합니다. 주의할 것은 마지막에 숫자를 적기 전에 스페이스를 하나 두는 것인데요, ~(틸데)를 그냥 쓰면 그 직전 명령이 이것을 잡아먹어서 나오지 않을 수가 있습니다. `{}~`라고 하여 스페이스를 살린 것을 유의하세요.

숙제에 대한 코멘트 모두가 `\int_if_odd:nTF`를 이용하는 와중에 신현 군이 재미있는 솔루션을 보내왔으므로 살펴보겠습니다.
`\l_tmpb_int`라는 카운터를 시작할 때 0으로 만들어 둡니다. 그리고 tl을 mapping하는 함수에서는

```
if l_tmpb_int == 0:
    l_tmpa_int = 1
```

```
else:
    l_tmpb_int = 0
```

이 과정을 반복하는 것입니다. `\l_tmpb_int == 0`이면 fbox하고 이 값이 1이면 색상 박스로 식자하게 하였습니다.
잘 생각해보면 이 기법은 옛날 boolean이 없는 언어에서 bool 대신 쓰던 것을 응용한 것임을 알 수 있습니다.
즉,

```
if tmpa == false:
    tmpa = true
else:
    tmpa = false
```

이렇게 하는 것과 동일할 테니, 이 아이디어를 다음과 같이 확장할 수 있습니다.

```
\ExplSyntaxOn
\NewDocumentCommand \shcmd { m }
{
    \bool_set_false:N \l_tmpa_bool
    \tl_map_function:nN { #1 } \sh_cmd_fn:n
}

\cs_new:Npn \sh_cmd_fn:n #1
{
    \bool_set_inverse:N \l_tmpa_bool

    \bool_if:NTF \l_tmpa_bool
    {
        \color_box_it:n { #1 }
    }
    {
        \f_box_it:n { #1 }
    }
}

\cs_new:Npn \color_box_it:n #1
{
    \colorbox {red } { \color{yellow } #1 }
}

\cs_new:Npn \f_box_it:n #1
{
    \fbox { #1 }
}
\ExplSyntaxOff

\shcmd{world}
```

연습문제

기본 1. 명령 `\acmd`는 두 개의 인자를 받는다. 첫 번째 인자는 숫자이며 두 번째 인자는 임의의 문자열이다. 만약 문자열이 지정된 숫자보다 크다면 앞에서부터 숫자에 해당되는 번째 문자까지만 출력하라. 만약 문자열이 지정된 숫자보다 작다면 문자열의 앞쪽에 _(언더스코어)를 붙여 n개의 문자열이 되도록 하라.

문자열이 긴 경우 카운터를 1씩 증가시켜가면서 mapping하면서, 이 카운터가 인자로 주어진 값 #1보다 커지면 아무것도 찍지 말고 그렇지 않으면 받은 토큰을 입력 스트림에 남기면 됩니다.

```
\ExplSyntaxOn

\cs_new:Npn \process_long:nn #1 #2
{
    \tl_set:Nn \l_tmpa_tl { #2 }
    \int_zero:N \l_tmpa_int

    \tl_map_inline:Nn \l_tmpa_tl
    {
        \int_incr:N \l_tmpa_int
        \int_compare:nTF { \l_tmpa_int > #1 }
        {  }
        { ##1 }
    }
}

%%% test
\process_long:nn { 5 } { beautiful }

\ExplSyntaxOff
```

beaut

빈 괄호를 그냥 두는 것이 보기 싫으면

```
\int_compare:nT { \l_tmpa_int <= #1 }
{ ##1 }
```

이렇게 해도 좋고요.
tl map은 언제라도 중단할 수 있으니, 다음처럼 해도 뭐.....

```
\ExplSyntaxOn

\cs_set:Npn \process_long:nn #1 #2
{
    \int_zero:N \l_tmpa_int
    \tl_map_inline:nn { #2 }
    {
        \int_incr:N \l_tmpa_int
        ##1
        \int_compare:nT { \l_tmpa_int >= #1 }
        {  \tl_map_break: }
    }
```

```
}

\process_long:nn { 5 } { beautiful }
\ExplSyntaxOff
```

beaut

이 예는 `\tl_map_break:`라는 게 있다는 걸 보이기 위한 것입니다. 위의 코드에서는 `##1`을 찍는 것이 먼저 있기 때문에 카운터가 `>=#1`이 되었을 때 중단해야 합니다. 만약 카운터 검사와 탈출 명령을 먼저 쓴다면 조건이 `>#1`이어야 할 테지요.
대체로 map 탈출 명령은 코드의 가독성을 떨어뜨리기 때문에 웬만하면 사용하지 않는 편이 좋습니다.

문자열이 준 숫자보다 짧을 경우 예를 들어 5와 abc가 주어지면 문자열 길이와의 차이 만큼을 뭔가로 메꾸면 되는 거지요.
이번 주의 학습 내용인 `\int_step_inline:`이나 `\int_do_while:`을 이미 알고 있다면 간단히 다음처럼 할 수 있습니다. 여기서는 `\int_step_inline:nn`을 썼습니다.

```
\ExplSyntaxOn
\cs_new:Npn \process_short:nn #1 #2
{
    \int_zero:N \l_tmpa_int
    \int_set:Nn \l_tmpb_int { #1 - \tl_count:n { #2 } }

    \tl_clear:N \l_tmpa_tl

    \int_step_inline:nn { \l_tmpb_int }
    {
        \tl_put_right:Nn \l_tmpa_tl { X }
    }

    \tl_put_right:Nn \l_tmpa_tl { #2 }

    \tl_use:N \l_tmpa_tl
}

\process_short:nn { 5 } { abc }
\ExplSyntaxOff
```

XXabc

`\l_tmpb_int`는 주어진 값과 문자열 길이의 차이입니다. 즉 추가해야 하는 글자 수에 해당하지요. 그 수만큼 X를 채우고 그 뒤에 `#2`를 붙였습니다.
이제 X 위치에 `\textunderscore`를 두면 해결됩니다. 여기서는 간단히 `#2`를 put right하는 방법으로 처리했는데, 원한다면 `\tl_concat:NNN` 같은 걸 쓸 수도 있겠지요.
이재호 군은 `\int_while_do:nn`을 이용하여 다음과 같이 해결하였습니다.

```
      {
        \int_zero:N \l_count_int
        \int_while_do:nn { \l_count_int < \l_tmpa_int - \l_tmpb_int }
          {
            _
```

```
            \int_incr:N \l_count_int
          }
        \l_tmpa_tl
      }
```

언더스코어 문자는 \textunderscore를 쓰는 것이 안전합니다. expl3 범위 밖으로 나가면 이 _ 부호는 "mathmode가 아니라"는 오류를 낼 수 있습니다.

그런데, 이 숙제를 하는 시점에서 \int_step_inline:nn 등을 몰랐다고 합시다. 방법이 없을까요? 채움 문자(아래 예에서는 X, 원래 문제에서 요구한 것은 _) 하나를 찍는 명령을 만들고 이 명령을 모자라는 수만큼 재귀적으로 반복시키면 다음과 같이 됩니다.

```
\ExplSyntaxOn
\cs_new:Npn \process_short_var:nn #1 #2
{
    \int_compare:nTF { \tl_count:n { #2 } >= #1 }
    {
        #2
    }
    {
        \process_short_var:nn { #1 } { X #2 }
    }
}

\process_short_var:nn { 5 } { ab }

\ExplSyntaxOff
```

XXXab

문제를 출제할 적에 염두에 둔 "모범답안"은 이 재귀함수를 만들어보는 것이었습니다만, 실용적으로 while이나 for를 쓰는 것이 훨씬 간편합니다.

종합 이 둘을 합치면 주어진 문제에 대한 답안을 작성할 수 있습니다. X는 \textunderscore로 바꿔주세요.

```
\ExplSyntaxOn
\NewDocumentCommand \acmd { m m }
{
    \int_compare:nTF { \tl_count:n { #2 } >= #1 }
    {
        \process_long:nn { #1 } { #2 }
    }
    {
        \process_short:nn { #1 } { #2 }
    }
}
\ExplSyntaxOff

\acmd{5}{beautiful}, \acmd{5}{abc}
```

beaut, XXabc

연습문제

발전 2. Python에는 문자열을 자르는(슬라이싱) 재미있는 기법이 있다. `\myslicing` 명령을 정의하되, 3개의 인자를 받아들이도록 하여 첫 인자로 주어지는 문자열을 #2부터 #3까지 슬라이싱하여 (즉 `mystring[`m`:`n`]`과 비슷) 출력하도록 하여라. 스페이스는 무시한다.

이 문제는 이재호 군의 답안을 소개하는 것으로 대신합니다.

```
\ExplSyntaxOn
\cs_new:Npn \slicing_fn:nnn #1 #2 #3
  {
    \tl_set:Nn \l_tmpa_tl { #1 }
    \int_zero:N \l_tmpa_int
    \tl_map_inline:Nn \l_tmpa_tl
      {
        \int_incr:N \l_tmpa_int
        \int_compare:nT { #2 <= \l_tmpa_int <= #3 }
          {
            ##1
          }
      }
  }

\NewDocumentCommand \myslicing { m m m }
  {
    \slicing_fn:nnn { #1 } { #2 } { #3 }
  }
\ExplSyntaxOff

\myslicing{Hello World}{3}{7}
```

lloWo

연습문제

발전 3. 새로운 명령 \myitemswap을 정의한다. 이 명령은 네 개의 인자를 받아들이며 첫 번째 인자가 문자열이다. 두 번째와 세 번째는 숫자인데, 주어지는 문자열의 아이템 번호들이다. 마지막 네 번째 인자는 임의의 매크로를 받는다. 주어진 문자열에서 #2번째 항목과 #3번째 항목을 교환(swap)하여 네 번째로 주어진 매크로에 넣어 반환하라. 숫자가 문자열의 범위를 벗어날 때의 에러처리 코드를 포함하라.

리스트의 n번째 아이템을 얻는 함수 \<type>_item:Nn라는 것을 알고 있다면 다음처럼 간단히 해결할 수 있습니다.

```
\ExplSyntaxOn

\NewDocumentCommand \myitemswapproto { m m m }
{
    \tl_set:No \l_a_tl { \tl_item:nn { #1 } { #2 } }
    \tl_set:No \l_b_tl { \tl_item:nn { #1 } { #3 } }

    \int_zero:N \l_tmpa_int
    \tl_map_inline:nn { #1 }
    {
        \int_incr:N \l_tmpa_int
        \int_compare:nTF { \l_tmpa_int == #2 }
        {
            \l_b_tl
        }
        {
            \int_compare:nTF { \l_tmpa_int == #3 }
            {
                \l_a_tl
            }
            {
                ##1
            }
        }
    }
}

\ExplSyntaxOff

\myitemswapproto{abcde}{2}{4}
```

adcbe

문제는 오류 처리 코드를 포함하라고 하고 있는데 이것은 다음 두 가지를 처리하면 됩니다.

1. #2와 #3의 크기가 역순일 때.
2. 입력되는 숫자가 #1 길이보다 큰 수일 때. 작은 쪽을 무조건 첫 번째 아이템으로, 큰 쪽은 마지막 아이템으로 처리합니다.

```
\int_new:N \l_l_int
\int_new:N \l_r_int
```

```
\NewDocumentCommand \myitemswap { m m m }
{
    \int_set:Nn \l_l_int { \int_min:nn { #2 } { #3 } }
    \int_set:Nn \l_r_int { \int_max:nn { #2 } { #3 } }

    \int_compare:nT { \l_l_int > \tl_count:n { #1 } }
    {
        \int_set:Nn \l_l_int { 1 }
    }
    \int_compare:nT { \l_r_int > \tl_count:n { #1 } }
    {
        \int_set:Nn \l_r_int { \tl_count:n { #1 } }
    }
....
```

이 둘을 합치면 되겠지요.

이번에는 \tl_item:Nn을 모르는 상황이라면 어떻게 해야 하는지 생각해봅니다. 다음 코드는 한 가지 방법인데 어떻게 된 것인지 잘 생각해봅시다. 핵심은 #2번째 아이템을 \l_a_tl에 넣고 #3번째 아이템을 \l_b_tl에 넣기만 하면 되는 겁니다. 에러 처리 코드는 우선 생략합니다. (이재호 군이 제출한 답이 이 방법을 사용하였습니다.)

```
\ExplSyntaxOn
\NewDocumentCommand \myitemswapb { mmm }
{
    \int_zero:N \l_tmpa_int

    \tl_map_inline:nn { #1 }
    {
        \int_incr:N \l_tmpa_int
        \int_compare:nTF { \l_tmpa_int == #2 }
        {
            \tl_set:Nn \l_a_tl { ##1 }
        }
        {
            \int_compare:nT { \l_tmpa_int == #3 }
            {
                \tl_set:Nn \l_b_tl { ##1 }
            }
        }
    }

    \int_zero:N \l_tmpa_int
    \tl_map_inline:nn { #1 }
    {
        \int_incr:N \l_tmpa_int
        \int_compare:nTF { \l_tmpa_int == #2 }
        {
            \l_b_tl
        }
        {
            \int_compare:nTF { \l_tmpa_int == #3 }
            {
```

```
                \l_a_tl
            }
            {
                ##1
            }
        }
    }
}
\ExplSyntaxOff
\myitemswapb{abcde}{2}{3}
```

acbde

한편, 이번 주에 학습하게 되는 `\int_case:nn` 문을 이용하면 코드를 더 간결하게 쓸 수 있지요. 이 모든 것을 모두 합친 코드를 보입니다.

```
\ExplSyntaxOn
\int_new:N \l_l_int
\int_new:N \l_r_int
\tl_new:N  \l_a_tl
\tl_new:N  \l_b_tl

\NewDocumentCommand \myitemswap { m m m }
{
    \int_set:Nn \l_l_int { \int_min:nn { #2 } { #3 } }
    \int_set:Nn \l_r_int { \int_max:nn { #2 } { #3 } }

    \int_compare:nT { \l_l_int > \tl_count:n { #1 } }
    {
        \int_set:Nn \l_l_int { 1 }
    }
    \int_compare:nT { \l_r_int > \tl_count:n { #1 } }
    {
        \int_set:Nn \l_r_int { \tl_count:n { #1 } }
    }

    \tl_set:No \l_a_tl { \tl_item:nn { #1 } { \l_l_int } }
    \tl_set:No \l_b_tl { \tl_item:nn { #1 } { \l_r_int } }

    \int_zero:N \l_tmpa_int
    \tl_map_inline:nn { #1 }
    {
        \int_incr:N \l_tmpa_int

        \int_case:nnF { \l_tmpa_int }
        {
            { \l_l_int } { \l_b_tl }
            { \l_r_int } { \l_a_tl }
        }
        {
            ##1
        }
    }
```

```
}

\ExplSyntaxOff
\myitemswap{abcde}{7}{2} \quad \myitemswap{abcde}{2}{5}
```

aecdb aecdb

에러 처리를 테스트하기 위하여 {7}{2} 순으로 인자를 주었습니다. {2}{5}와 같은 결과가 나와야 합니다.

연습문제

기본 1. 주어지는 문자열을 앞에서부터 3개째마다 쉼표를 추가하는 명령을 작성하여라.

입력: `\test{this is just a test}`
출력: thi, sis, jus, tat, est

문제의 핵심은 마지막에도 쉼표가 붙으면 이를 제거하라는 것이지요. 앞서 배운 어떤 방법으로든 일단 세 개마다 쉼표를 붙이는 문제는 해결했다고 보고요, 그러면 일단 처리해야 할 문자열은

```
thi, sis, jus, tat, est,
```

일 것입니다.

(1) clist를 이용하는 방법 변환된 문자열을 clist에 넣고 해결하는 게 매우 간단합니다. 빈 아이템을 삭제하는 것이 트릭의 핵심입니다.

```
\ExplSyntaxOn
\cs_new:Npn \remove_last_comma:n #1
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \clist_set:NV \l_tmpa_clist \l_tmpa_tl
    \clist_remove_all:Nn \l_tmpa_clist {}

    \clist_use:Nn \l_tmpa_clist {,~}
}

\remove_last_comma:n { thi,sis,jus,tst,est, }

\ExplSyntaxOff
```

thi, sis, jus, tst, est

(2) tl 길이 쉼표를 붙이기 전에 tl의 길이를 먼저 재어서, 만약 3의 배수이면 bool 변수 하나를 true로 해두었다가 이를 나중에 조작할 수 있습니다. 다음 코드에서 `\regex...`를 쓴 부분은 세 개마다 쉼표를 붙이는 부분인데 다른 방법으로 해도 상관없습니다. 예시 코드가 너무 길어지는 것이 거시기해서 이걸 쓴 것뿐입니다.

```
\ExplSyntaxOn
\cs_new:Npn \test_fn:n #1
{
    \tl_set:Nn \l_tmpa_tl { #1 }

    \int_compare:nTF { \int_mod:nn { \tl_count:n { #1 } } { 3 } == 0 }
    {
        \bool_set_true:N \l_tmpa_bool
    }
    {
        \bool_set_false:N \l_tmpa_bool
    }

    \regex_replace_all:nnN { (.)(.)(.) } { \1\2\3, } \l_tmpa_tl
```

```
    \bool_if:NTF \l_tmpa_bool
    {
        \tl_reverse:N \l_tmpa_tl
        \tl_set:No \l_tmpb_tl { \tl_tail:N \l_tmpa_tl }
        \tl_reverse:N \l_tmpb_tl
        \l_tmpb_tl
    }
    {
        \l_tmpa_tl
    }

}

\test_fn:n { This is just a test }

\ExplSyntaxOff
```

```
,tse,tat,suj,sis,ihT
```

(3) 인덱스 카운터를 이용한 마지막 아이템 처리 tl에 대하여 인덱스 카운터를 주어서 map하다가 마지막 아이템이면 이를 출력하지 않는 방법이 있습니다. 가장 이해하기 쉽고 많이 쓰이는 방법입니다.

```
\ExplSyntaxOn

\cs_new:Npn \test_two:n #1
{
    \int_zero:N \l_tmpa_int
    \tl_clear:N \l_tmpb_tl

    \tl_map_inline:nn { #1 }
    {
        \int_incr:N \l_tmpa_int

        \int_compare:nTF { \l_tmpa_int == \tl_count:n { #1 } }
        {
            \str_if_eq:nnTF { ##1 } { , }
            {
                % do nothing
            }
            {
                \tl_put_right:Nn \l_tmpb_tl { ##1 }
            }
        }
        {
            \tl_put_right:Nn \l_tmpb_tl { ##1 }
        }
    }

    \l_tmpb_tl
}
```

```
\test_two:n { thi,sis,jus,tat,est, }

\ExplSyntaxOff
```

thi,sis,jus,tat,est

이것을 다시 생각하면, 아예 처음에 쉼표를 붙여 분리할 때 인덱스 카운터가 마지막 아이템을 가리키고 있으면 쉼표를 안 붙이게 만들 수 있을 것입니다. 그렇게 하면 대략 다음과 같이 됩니다.

```
\ExplSyntaxOn

\NewDocumentCommand \test { m }
{
    \int_zero:N \l_tmpa_int
    \tl_clear:N \l_tmpb_tl

    \tl_map_inline:nn { #1 }
    {
        \int_incr:N \l_tmpa_int
        \int_compare:nTF { \int_mod:nn { \l_tmpa_int } { 3 } == 0 }
        {
            \tl_put_right:Nn \l_tmpb_tl { ##1 }
            \int_compare:nTF { \l_tmpa_int == \tl_count:n { #1 } }
            { }
            {
                \tl_put_right:Nn \l_tmpb_tl { ,~ }
            }
        }
        {
            \tl_put_right:Nn \l_tmpb_tl { ##1 }
        }
    }

    \l_tmpb_tl
}

\ExplSyntaxOff

\test{This is just a test}
```

Thi, sis, jus, tat, est

이게 원래 문제를 출제할 때 생각한 정답에 가깝습니다.

(4) regex 방법 마지막 토큰이 쉼표면 이를 제거하라는 거는 한 줄이면 됩니다. “재미삼아” 콤파 다음에 스페이스 하나씩 주는 명령 한 줄을 더 추가했습니다.

```
\ExplSyntaxOn
\cs_set:Npn \remove_last_comma:n #1
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \regex_replace_once:nnN { \, $ } { } \l_tmpa_tl
```

```
    \regex_replace_all:nnN { \, } { \, \  } \l_tmpa_tl
    \tl_use:N \l_tmpa_tl
}

\remove_last_comma:n { thi,sis,jus,tst,est, }

\ExplSyntaxOff
```

thi, sis, jus, tst, est

문제 전체를 regex로 해결하면,

```
\ExplSyntaxOn

\NewDocumentCommand \testa { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \regex_replace_all:nnN { \s } { } \l_tmpa_tl   %% remove all horiz spaces
    \regex_replace_all:nnN { (.)(.)(.) } { \1\2\3,\ } \l_tmpa_tl
    \regex_replace_once:nnN { \, \s $ } { } \l_tmpa_tl
    \l_tmpa_tl
}

\ExplSyntaxOff

\testa{This is just a test}
```

Thi, sis, jus, tat, est

연습문제

기본 1. 새로운 명령 `\newcmd`는 다음과 같은 형식으로 실행한다.
`\newcmd{this is just a test}`
주어지는 인자를 먼저 세 개마다 쉼표를 붙여 분리하고, 분리된 각 단어를 `\resi`, `\resii`, `\resiii`, `\resiv`, ...에 넣어 반환하라.

쉼표로 분리하는 것은 이미 되었다고 하겠습니다. 이제 문제가 되는 것은 분리된 각 단어를 매크로에 넣어 반환하는 정도겠네요. 가장 쉬운 게 clist입니다.

```
\ExplSyntaxOn
\NewDocumentCommand \newcmdproto { m }
{
    \clist_set:Nn \l_tmpa_clist { #1 }
    \int_zero:N \l_tmpa_int
    \clist_map_inline:Nn \l_tmpa_clist
    {
        \int_incr:N \l_tmpa_int
        \tl_set:cn { res \int_to_roman:n { \l_tmpa_int } } { ##1 }
    }
}
\ExplSyntaxOff
\newcmdproto{thi,isi,jus,tat,est}
\resi, \resii, \resiv
```

thi, isi, tat

연습문제

발전 2. 인자로 주어지는 단어의 각 문자가 몇 번씩 사용되었는지를 예시와 같이 출력하여라.

소팅 접근 입력된 문자열을 소팅해서 mapping하다가, 앞서와 다른 문자가 나오면 출력하는 방법으로 처리해봅니다.
토큰 리스트 tl의 소팅은 개별 문자 비교로 다음과 같이 할 수 있습니다.

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { minuet }
\tl_sort:Nn \l_tmpa_tl
{
    \int_compare:nTF { `#1 > `#2 }
    { \sort_return_swapped: }
    { \sort_return_same: }
}
\l_tmpa_tl
\ExplSyntaxOff
```

eimntu

그런데 만약 비교해야 하는 것이 개별 문자가 아니라 문자열이라면

```
\ExplSyntaxOn
\clist_set:Nn \l_tmpa_clist { tomato, apple, grape, banana, kiwi }
\clist_sort:Nn \l_tmpa_clist
{
    \int_compare:nTF { \pdftex_strcmp:D { #1 } { #2 } > 0 }
    { \sort_return_swapped: }
    { \sort_return_same: }
}
\clist_use:Nn \l_tmpa_clist {,~}
\ExplSyntaxOff
```

apple, banana, grape, kiwi, tomato

이렇게 하라고 배웠습니다. 이제 `\esg_str_cmp:nn`이라는 명령을 하나 정의해둡시다. 이 명령은 앞으로도 문자열 소팅을 할 적에는 `\pdftex_strcmp:D` 대신 사용합니다.

```
\ExplSyntaxOn
\cs_new:Npn \esg_str_cmp:nn #1 #2
{
    \sys_if_engine_luatex:TF
    {
        \directlua { l3kernel.strcmp ( '#1', '#2' ) }
    }
    {
        \pdftex_strcmp:D { #1 } { #2 }
    }
}

\cs_generate_variant:Nn \esg_str_cmp:nn { Vn }
```

```
%%% test
\esg_str_cmp:nn { ab } { bc } \quad
\esg_str_cmp:nn { ab } { ab } \quad
\esg_str_cmp:nn { bcd } { abc }
\ExplSyntaxOff
```

```
-1  0  1
```

인자를 확장해야 할 때를 위해서 Vn variant를 준비했습니다. 아래 사용례가 있으니 참고하세요. LuaTEX에서도 동작합니다.

문제를 다음과 같이 해결합니다.

(1) 입력문자열(여기서는 abracadabra)을 sorting하여 aaaaabbcdrr로 만듭니다.

(2) 이 문자열을 mapping하면서, 첫 번째 아이템 'a'가 들어올 때,

  (a) 이전 문자를 저장할 `\l_prev_tl`에 이 문자를 넣어둡니다.

  (b) 현재 문자의 카운터 `\l_icnt_int`를 0으로 합니다.

(3) 그 다음 글자부터

  (a) 현재 아이템이 `\l_prev_tl`과 같은 글자인지 검사

  (b) 만약 같으면 `\l_icnt_int`를 1 증가합니다.

  (c) 만약 다르면, "현재 글자 = 현재 icnt 카운터"를 출력하고, `\l_icnt_int`를 0으로, `\l_prev_tl`을 현재 글자로 설정합니다.

(4) 마지막 글자에 대하여 처리가 필요합니다. 마지막 글자가 이전 글자와 다르다면 카운터가 0일 것이므로 이를 1로 만들어야 합니다. 만약 같은 글자라면 역시 카운터를 1 증가시켜야 합니다.

하나씩 차근차근 따라가보겠습니다. 먼저 사용자 변수를 설정.

```
\tl_new:N \l_prev_tl
\int_new:N \l_icnt_int
```

소팅.

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { abracadabra }
\tl_sort:Nn \l_tmpa_tl
{
    \int_compare:nTF { `#1 > `#2 }
    { \sort_return_swapped: } { \sort_return_same: }
}

%%% test
\tl_use:N \l_tmpa_tl
\ExplSyntaxOff
```

```
aaaaabbcdrr
```

소팅된 tl을 mapping합니다. 카운터를 위해 `\l_tmpa_int`를 사용합니다.

```
\int_zero:N \l_tmpa_int

\tl_map_inline:Nn \l_tmpa_tl
{
    \int_incr:N \l_tmpa_int
```

인덱스 카운터가 1일 때

```
    \int_compare:nTF { \l_tmpa_int == 1 }
    {
        \int_zero:N \l_icnt_int
        \tl_set:Nn \l_prev_tl { #1 }
    }
```

1이 아니면 현재 아이템이 이전 문자와 같은지를 검사합니다.

```
    {
        \str_if_eq:eeTF { \l_prev_tl } { #1 }
```

문자 매크로와 문자의 동일성을 검사하는 데 여기서는 `\str_if_eq:ee`를 썼는데, 다음과 같이 해도 같은 결과입니다.

```
\int_compare:nTF { \expandafter `\l_prev_tl == `#1 }
```

여기서는 `\expandafter`를 써야 합니다. 이 매크로를 확장해야 하기 때문입니다. 이번 주에 배우는 expl3 확장명령을 써서

```
\int_compare:nTF { \exp_args:No `\l_prev_tl == `#1 }
```

이것도 좋고, 아까 정의한 string compare 명령을 써서

```
\int_compare:nTF { \esg_str_cmp:Vn \l_prev_tl { #1 } == 0 }
```

이렇게도 가능합니다. 다만 `:Vn` 인자확장 지시는 우리가 이미 정의해서 제공하기 때문에 쓸 수 있는 것입니다.
이 검사가 참일 때, 즉 현재 아이템이 이전 문자와 같을 때는

```
            {
                \int_incr:N \l_icnt_int
            }
```

다를 때는

```
            {
                \int_incr:N \l_icnt_int
                \l_prev_tl = \int_use:N \l_icnt_int \par
                \int_zero:N \l_icnt_int
                \tl_set:Nn \l_prev_tl { #1 }
            }
```

만약 현재 아이템이 tl의 마지막 아이템이라면 다음과 같이 처리합니다.

```
            \int_compare:nT { \l_tmpa_int == \tl_count:N \l_tmpa_tl }
            {
                \int_incr:N \l_icnt_int
                \l_prev_tl = \int_use:N \l_icnt_int \par
            }
```

여기까지입니다.

```
     }  %% end of int_compare_false
}  %% end of map
```

이것을 실행하면 다음과 같다.

```
\ExplSyntaxOn
\int_new:N \l_icnt_int
\tl_new:N \l_prev_tl

\tl_set:Nn \l_tmpa_tl { abracadabra }
\tl_sort:Nn \l_tmpa_tl
{
    \int_compare:nTF { `#1 > `#2 }
    { \sort_return_swapped: } { \sort_return_same: }
}
\int_zero:N \l_tmpa_int

\tl_map_inline:Nn \l_tmpa_tl
{
    \int_incr:N \l_tmpa_int
    \int_compare:nTF { \l_tmpa_int == 1 }
    {
        \int_zero:N \l_icnt_int
        \tl_set:Nn \l_prev_tl { #1 }
    }
    {
        \str_if_eq:eeTF { \l_prev_tl } { #1 }
            {
                \int_incr:N \l_icnt_int
            }
            {
                \int_incr:N \l_icnt_int
                \l_prev_tl = \int_use:N \l_icnt_int \par
                \int_zero:N \l_icnt_int
                \tl_set:Nn \l_prev_tl { #1 }
            }
            \int_compare:nT { \l_tmpa_int == \tl_count:N \l_tmpa_tl }
            {
                \int_incr:N \l_icnt_int
                \l_prev_tl = \int_use:N \l_icnt_int \par
            }
     }  %% end of int_compare_false
}  %% end of map
\ExplSyntaxOff
```

```
a=5
b=2
c=1
d=1
r=2
```

이것이 잘 되는 것을 확인하였으면 이상의 코드를 함수 형태로 작성합니다. 주의할 점은 함수 안에서가 아니라 위에서와 같이 그냥 테스트할 때 #1이라 한 것(inline map에서의 현재 item)은 함수 정의 안에 들어가게 되면 ##1이 되어야 한다는 것입니다. 함수 형태로 구현하는 코드는 사실상 위의 것과 동일하므로 다시 반복하지 않겠습니다.
이재호 군은 `\pdftex_strcmp:D`를 이용하여 구현했는데 잘 했습니다.

prop 접근 property list (prop)를 이용하여 해결하는 예를 설명없이 보이겠습니다.

```
\ExplSyntaxOn
\NewDocumentCommand \testcmdp { m }
{
    \prop_clear:N \l_tmpa_prop

    \tl_map_inline:nn { #1 }
    {
        \prop_if_in:NnTF \l_tmpa_prop { ##1 }
        {
            \prop_get:NnN \l_tmpa_prop { ##1 } \l_tmpa_tl
            \int_set:Nn \l_tmpa_int { \l_tmpa_tl + 1 }
            \prop_put:NnV \l_tmpa_prop { ##1 } \l_tmpa_int
        }
        {
            \prop_put:Nnn \l_tmpa_prop { ##1 } { 1 }
        }
    }

    \clist_clear:N \l_tmpa_clist

    \prop_map_inline:Nn \l_tmpa_prop
    {
        \clist_put_right:Nn \l_tmpa_clist { ##1 ~=~ ##2 }
    }

    \clist_sort:Nn \l_tmpa_clist
    {
        \int_compare:nTF { \esg_str_cmp:nn { ##1 } { ##2 } > 0 }
        { \sort_return_swapped: }
        { \sort_return_same: }
    }

    \clist_use:Nn \l_tmpa_clist { \par }

}
\ExplSyntaxOff

\testcmdp{abracadabra}
```

```
a = 5
b = 2
c = 1
d = 1
r = 2
```

regex + 재귀 접근 다음처럼 하겠다는 것입니다.

```
import re

def regexcnt(string,pattern):
    return len(re.findall(pattern,string))

def myfunc(s):
    a=s[0]
    b=regexcnt(s,s[0])
    print(a,"=",b)
    s=s.replace(s[0],"")
    if len(s) != 0:
        myfunc(s)

myfunc('abracadabra')
```

이것을 expl3로 씁니다.

```
\ExplSyntaxOn

\NewDocumentCommand \testcmdr { m }
{
    \test_cmd_r:n { #1 }
}

\cs_new:Npn \test_cmd_r:n #1
{
    \tl_set:Nn \l_tmpa_tl { #1 }

    \tl_set:No \l_tmpb_tl { \tl_head:N \l_tmpa_tl }
    \exp_args:Nxx \regex_count:nnN { \l_tmpb_tl } { \l_tmpa_tl } \l_tmpa_int

    \l_tmpb_tl {}~=~ \int_use:N \l_tmpa_int \par

    \exp_args:NNx \tl_remove_all:Nn \l_tmpa_tl { \l_tmpb_tl }

    \int_compare:nT { \tl_count:N \l_tmpa_tl != 0 }
    {
       \exp_args:Nx \test_cmd_r:n { \l_tmpa_tl }
    }
}
```

```
\ExplSyntaxOff

\testcmdr{abracadabra}
```

```
a = 5
b = 2
r = 2
c = 1
d = 1
```

결과가 소팅이 안 되어 있네요. 알파벳 순으로 결과를 나열하고 싶다면 clist를 하나 비운 다음에 위의 출력하는 부분 (`\par`가 있는 행)을 그냥 출력하지 말고 clist의 아이템으로 `\clist_put_right:Nx`한 다음 마지막에 이 clist를 sort하고 `\clist_use:Nn`하면 됩니다. 그 부분은 관심있으면 직접 해보시길.

Nova de Hi.