예제로 배우는 expl3

정답과 해설 모음

이엑스피엘쓰리 스타디 그룹

2019/08/10, 2022/05/12. version 3.0

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] □□□군의 해결책 1에 관하여 한편, □□□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
```

w o r l d

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
```

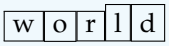
a, bc
b, c

`\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}
```



문제를 해결하는 아이디어는 □□□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 □□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 \acmd는 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 ##1을 찍는 것이 먼저 있기 때문에 카운터가 >=#1이 되었을 때 중단해야 합니다. 만약 카운터 검사와 탈출 명령을 먼저 쓴다면 조건이 >#1이어야 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 #2를 붙였습니다.
이제 X 위치에 `\textunderscore`를 두면 해결됩니다. 여기서는 간단히 #2를 put right하는 방법으로 처리했는데, 원한다면 `\tl_concat:NNN` 같은 걸 쓸 수도 있겠지요.
□□□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 □□□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

참고로 interface3에는 `\tl_range:nnn`이라는 함수를 제공하고 있습니다. 우리가 정의한 이 함수와 하는 일이 거의 같습니다.

```
\ExplSyntaxOn
\tl_range:nnn { Hello World } { 3 } { 7 }
\ExplSyntaxOff
```

lloWo

이 함수는 space를 처리하기 위한 (비교적) 복잡한 내부 처리 과정을 가지고 있습니다. 그리고 첫 인자가 macro가 아닙니다. 그러므로 매크로를 사용할 적에는 그것을 한 번 확장해주어야 합니다.

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { Hello~World }
\exp_args:No \tl_range:nnn { \l_tmpa_tl } { 3 } { 7 }
\ExplSyntaxOff
```

llo Wo

연습문제

발전 3. 새로운 명령 `\myitemswap`을 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:Nn`라는 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.

```
\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 \newcmd는 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 { \tex_strcmp:D { #1 } { #2 } > 0 }
    { \sort_return_swapped: }
    { \sort_return_same: }
}
\clist_use:Nn \l_tmpa_clist {,~}
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_new:Npn \esg_str_cmp:nn #1 #2
{
    \sys_if_engine_luatex:TF
    {
        \directlua { l3kernel.strcmp ( '#1', '#2' ) }
    }
    {
        \tex_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_tl에 이 문자를 넣어둡니다.

  (b) 현재 문자의 카운터 \l_icnt_int를 0으로 합니다.

(3) 그 다음 글자부터

  (a) 현재 아이템이 \l_prev_tl과 같은 글자인지 검사

  (b) 만약 같으면 \l_icnt_int를 1 증가합니다.

  (c) 만약 다르면, "현재 글자 = 현재 icnt 카운터"를 출력하고, \l_icnt_int를 0으로, \l_prev_tl을 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_int를 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 같습니다.

```
\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. (입력되는 문자열이 단 한 글자일 때의 오류처리를 만들어두면 더 좋습니다.)
□□□군은 `\tex_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 (\par가 있는 행)을 그냥 출력하지 말고 clist의 아이템으로 \clist_put_right:Nx한 다음 마지막에 이 clist를 sort하고 \clist_use:Nn하면 됩니다. 그 부분은 관심있으면 직접 해보시길.

연습문제

응용 1. 다음과 같이 이루어진 수열이 있다. 인자로 주어지는 n번째 항까지의 합을 출력하여라.

3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84, 2, 1, 1, 15, 3, 13, 1, 4, 2, 6, 6, 99, 1, 2, 2, 6, 3, 5, 1, 1, 6, 8, 1, 7, 1, 2, 3, 7, 1, 2, 1, 1, 12, 1, 1, 1, 3, 1, 1, 8, 1, 1, 2, 1, 6, 1, 1, 5, 2, 2, 3, 1, 2, 4, 4, 16, 1, 161, 45, 1, 22, 1, 2, 2, 1, 4, 1, 2, 24, 1, 2, 1, 3, 1, 2

seq와 clist clist가 seq의 간략화한 것임을 이미 언급하였습니다. 일반적으로 seq가 속도도 (극히 조금) 빠르고 할 수 있는 일도 많습니다. 그러므로 "리스트"가 문제가 되면 대체로 seq로 문제를 해결하는 것이 좋습니다.
그러면 clist라는 것은 왜 있는가? LaTeX에 콤마로 분리된 리스트가 많이 쓰였기 때문입니다. 특히 명령이나 패키지 선언의 인자로 이런 형태가 자주 나오는데 이것을 간편하게 처리하기 위한 장치입니다.
지금까지 clist는 충분히 연습했으므로 이 문제는 seq로 해결해봅니다.

```
\ExplSyntaxOn
\seq_set_from_clist:Nn \l_tmpa_seq
{
    3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84, 2,
     ↪ 1, 1, 15, 3, 13, 1, 4, 2, 6, 6, 99, 1, 2, 2, 6, 3, 5, 1, 1, 6, 8, 1,
     ↪ 7, 1, 2, 3, 7, 1, 2, 1, 1, 12, 1, 1, 1, 3, 1, 1, 8, 1, 1, 2, 1, 6, 1,
     ↪ 1, 5, 2, 2, 3, 1, 2, 4, 4, 16, 1, 161, 45, 1, 22, 1, 2, 2, 1, 4, 1,
     ↪ 2, 24, 1, 2, 1, 3, 1, 2
}
\ExplSyntaxOff
```

clist에서는 item의 index를 처리하기 위하여 int 하나를 zero로 만들고 map function에서 이것을 incr하면서 세었습니다. 이것도 나쁘지 않지만 seq에는 아예 인덱스를 활용할 수 있는 함수가 제공됩니다. 세 개마다 fbox하는 일을 해보자면,

```
\ExplSyntaxOn
\seq_set_from_clist:Nn \l_tmpa_seq
{
    3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84, 2,
     ↪ 1, 1, 15, 3, 13, 1, 4, 2, 6, 6, 99, 1, 2, 2, 6, 3, 5, 1, 1, 6, 8, 1,
     ↪ 7, 1, 2, 3, 7, 1, 2, 1, 1, 12, 1, 1, 1, 3, 1, 1, 8, 1, 1, 2, 1, 6, 1,
     ↪ 1, 5, 2, 2, 3, 1, 2, 4, 4, 16, 1, 161, 45, 1, 22, 1, 2, 2, 1, 4, 1,
     ↪ 2, 24, 1, 2, 1, 3, 1, 2
}

\seq_indexed_map_inline:Nn \l_tmpa_seq
{
    \int_compare:nTF { \int_mod:nn { #1 } { 3 } == 0 }
    {
        \fbox { #2 }
    }
    {
        #2
    }
```

```
    \int_compare:nF { \seq_count:N \l_tmpa_seq == #1 }
    {
        ,~
    }
}
\ExplSyntaxOff
```

3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84, 2, 1, 1, 15, 3, 13, 1, 4, 2, 6, 6, 99, 1, 2, 2, 6, 3, 5, 1, 1, 6, 8, 1, 7, 1, 2, 3, 7, 1, 2, 1, 1, 12, 1, 1, 1, 3, 1, 1, 8, 1, 1, 2, 1, 6, 1, 1, 5, 2, 2, 3, 1, 2, 4, 4, 16, 1, 161, 45, 1, 22, 1, 2, 2, 1, 4, 1, 2, 24, 1, 2, 1, 3, 1, 2

`\seq_indexed_map_function:`을 쓸 적에 주의할 점은 인덱스가 #1 (##1), 아이템이 #2 (##2)라는 것입니다. 착각하지 않도록 하세요.

문제 자체는 어렵지 않습니다. 문제에서 주어진 수열은 "고정된" 것으로서 이 수열 자체에 조작을 가할 일이 없으므로 "상수"로 선언할 수 있습니다. 그리고 합을 구하는 과정은 전역 변수 `\g_sum_int`를 선언해두고 여기에 값을 넣어 전달하는 방식(procedure)으로 처리하겠습니다.

```
\ExplSyntaxOn
\seq_const_from_clist:Nn \c_picontfrac_seq
{
    3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84, 2,
     ↪ 1, 1, 15, 3, 13, 1, 4, 2, 6, 6, 99, 1, 2, 2, 6, 3, 5, 1, 1, 6, 8, 1,
     ↪ 7, 1, 2, 3, 7, 1, 2, 1, 1, 12, 1, 1, 1, 3, 1, 1, 8, 1, 1, 2, 1, 6, 1,
     ↪ 1, 5, 2, 2, 3, 1, 2, 4, 4, 16, 1, 161, 45, 1, 22, 1, 2, 2, 1, 4, 1,
     ↪ 2, 24, 1, 2, 1, 3, 1, 2
}

\int_new:N \g_sum_int

\NewDocumentCommand \testsum { m }
{
    \int_zero:N \g_sum_int

    \int_compare:nTF { #1 > \seq_count:N \c_picontfrac_seq }
    {
        number~too~large. ($\le$ \seq_count:N \c_picontfrac_seq ) \par
    }
    {
        \test_sum:n { #1 }
        \int_use:N \g_sum_int
    }
}

\cs_new:Npn \test_sum:n #1
{
    \seq_indexed_map_inline:Nn \c_picontfrac_seq
    {
        \int_compare:nTF { ##1 <= #1 }
        {
            \int_gadd:Nn \g_sum_int { ##2 }
        }
        {
            \seq_map_break:
        }
    }
}
```

```
\testsum{5}, \quad \testsum{500}

\ExplSyntaxOff
```

318, number too large.(≤97)

intarray 한편, 정수로 이루어진 리스트 자료형으로 intarray라는 것이 있습니다. 이름 그대로 정수 배열이기는 한데 일반적인 리스트처럼 mapping하거나 조작하는 것은 할 수 없고 그 대신 각 아이템에 대한 접근 효율을 높인 것이라고 보면 됩니다. (seq의 아이템에 대한 접근 시간은 linear인 반면 intarray는 constant) 주어진 문제는 이것을 이용하는 것이 가능합니다.

```
\ExplSyntaxOn
\intarray_const_from_clist:Nn \l_tmpa_intarray
{
    3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84, 2,
     ↪ 1, 1, 15, 3, 13, 1, 4, 2, 6, 6, 99, 1, 2, 2, 6, 3, 5, 1, 1, 6, 8, 1,
     ↪ 7, 1, 2, 3, 7, 1, 2, 1, 1, 12, 1, 1, 1, 3, 1, 1, 8, 1, 1, 2, 1, 6, 1,
     ↪ 1, 5, 2, 2, 3, 1, 2, 4, 4, 16, 1, 161, 45, 1, 22, 1, 2, 2, 1, 4, 1,
     ↪ 2, 24, 1, 2, 1, 3, 1, 2
}

\NewDocumentCommand \testsuma { m }
{
    \int_zero_new:N \g_sum_int

    \int_compare:nTF { #1 > \intarray_count:N \l_tmpa_intarray }
    {
        number~too~large. ($\le$ \intarray_count:N \l_tmpa_intarray ) \par
    }
    {
        \test_suma:n { #1 }
        \int_use:N \g_sum_int
    }
}

\cs_new:Npn \test_suma:n #1
{
    \int_step_inline:nn { #1 }
    {
        \int_gadd:Nn \g_sum_int
            { \intarray_item:Nn \l_tmpa_intarray { ##1 } }
    }
}

\ExplSyntaxOff

\testsuma{5}
```

318

정수로 이루어진 리스트의 경우에 `\seq_item:Nn`보다 `\intarray_item:Nn`이 조금 효율이 높습니다.

숙제에 대한 코멘트

□□□군은 clist에 넣고 `\int_step_inline:nn` 하면서 `\clist_item:Nn`으로 취한 값을 더해가는 방

법을 썼습니다. 표준적인 해결방법이고 다음 한 줄이 핵심입니다.

```
\int_add:Nn \l_tmpa_int { \clist_item:Nn \l_my_clist { ##1 } }
```

□□□군은 이 방법이 뭔가 (뻔해서?) 마음에 들지 않았는지 `\clist_map_inline:Nn`으로 처리하다가 `\clist_map_break:`하는 방법과 `\clist_pop:NN`을 이용하는 방법을 보여주었습니다.
다음은 `\clist_pop:NN`을 이용한 코드입니다.

```
\int_zero:N \l_tmpa_int
\int_step_inline:nn { #1 }
  {
    \clist_pop:NN \l_tmpa_clist \l_tmpa_tl
    \int_add:Nn \l_tmpa_int { \l_tmpa_tl }
  }
\int_use:N \l_tmpa_int
```

`\clist_pop:NN`은 원본 clist에 변경을 가합니다. 이 점을 유념하여 clist의 재사용이 문제되지 않는 상황이라면 좋은 해결책입니다.

연습문제

기본 2. 주어진 수까지 더하는 대신 곱한다면 계승(factorial)을 구하는 trivial한 함수를 만들 수 있다. 이를 작성하여라. 단 인자는 12이하의 정수로 한다.

```
\ExplSyntaxOn
\NewDocumentCommand \simplefact { m }
{
    \int_set:Nn \l_tmpa_int { \c_one_int }

    \int_step_inline:nn { #1 }
    {
        \int_set:Nn \l_tmpa_int { \l_tmpa_int * ##1 }
    }

    \int_use:N \l_tmpa_int
}
\ExplSyntaxOff

\simplefact{10}
```

3628800

`\int_set:Nn`은 왜 `\int_set:Nx`나 `\int_set:No`가 없는가에 대해 생각해봅시다.
interface 함수 `\int_set:Nn`의 두 번째 인자 n 부분은 사실 `\int_eval:n`을 거친 결과를 받아들입니다. 이것을 "정수 표현식"이라고 하는데 `\int_set:Nn`의 두 번째 인자는 이 "정수 표현식"이 입력되어야 하는 것입니다.
`\int_eval:n`은 그 인자로 오는 정수 표현식 안에 있는 tl을 x-확장한 결과가 other 숫자이면 유효하게 처리합니다. 그리고 이미 int로 정의된 변수라면 이것을 확장하여 정수로 인식합니다. 즉, 다음과 같은 식은 효력이 있습니다.

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int { 10 }
\tl_set:Nn \l_tmpa_tl { 100 }
\int_set:Nn \l_tmpb_int { \l_tmpa_int + \l_tmpa_tl }
\int_use:N \l_tmpb_int
\ExplSyntaxOff
```

110

세 번째 줄 `\int_set:Nn \l_tmpb_int` 부분의 인자는 사실 모두 확장되어야 할 매크로를 포함하지만 이 인자 부분이 미리 `\int_eval:n` 처리를 거치므로 별도의 확장 명령을 쓰지 않아도 모두 처리되는 것입니다. 그러므로, □□□군이

```
\exp_args:NNo \int_set:Nn \l_tmpa_int { \l_tmpa_tl }
```

처럼 한 것이나, □□□군이

```
\int_set:Nn \l_tmpa_int { \int_eval:n { \l_tmpa_tl } }
```

한 것은, (지금까지 강의를 잘 따라온 증거겠지만) 간단히

```
\int_set:Nn \l_tmpa_int { \l_tmpa_tl }
```

로 충분하다는 것을 알 수 있습니다.

`\clist_item:Nn`이나 `\seq_item:Nn`의 n인자와 같이 "반드시 정수"를 요구하는 interface 명령은 대부분 `\int_eval:n`을 거치도록 정의되어 있습니다.
그러나, 사용자 정의 명령이라면 정수 표현식의 처리는 함수 작성자에게 그 책임이 있습니다.
또한,

```
\int_set:Nn \l_tmpa_int { \l_tmpa_int * 10 }
```

이런 변수 할당 식에서 `\int_eval:n`의 효과 때문에 먼저 괄호 안이 확장되고 그 다음에 할당이 이루어집니다.

예시 답안에 `\c_one_int`라는 상수 하나를 썼습니다. 숫자 상수는 `\c_zero_int`와 `\c_one_int`, 그리고 `\c_max_int`가 있는데 각각 다음 수를 의미합니다.

```
\ExplSyntaxOn
\int_use:N \c_zero_int \quad
\int_use:N \c_one_int \quad
\int_use:N \c_max_int
\ExplSyntaxOff
```

```
0   1   2147483647
```

상수를 쓰면 0이나 1의 경우에 parsing하는 데 걸리는 시간이 절약된다고 합니다만 큰 차이 있겠어요?

확장 문제의 보충 다음 예를 보세요.

```
\ExplSyntaxOn
\clist_set:Nn \l_tmpa_clist { 3, 1, 4, 1, 5, 9, 2 }

\int_zero:N \l_tmpa_int
\tl_clear:N \l_tmpa_tl

\cs_new:Npn \test_cs:n #1
{
    \int_step_inline:nn { #1 }
    {
        \int_set:Nn \l_tmpb_int { ##1 }
        \int_add:Nn \l_tmpa_int { \clist_item:Nn \l_tmpa_clist { \l_tmpb_int
        ↪  + 1 } }
        \tl_put_right:Nn \l_tmpa_tl { \clist_item:Nn \l_tmpa_clist {
        ↪  \l_tmpb_int + 1 } }
    }
}

\test_cs:n { 4 }
\int_use:N \l_tmpa_int \\
\tl_use:N \l_tmpa_tl
\ExplSyntaxOff
```

```
11
5555
```

둘 다 `:Nn`을 썼는데, 위의 문장은 성공적으로 아이템을 가져오지만 뒤의 것은 "맨 마지막에 호출된 아이템"만이 성공합니다. 그 이유를 설명할 수 있겠나요? 이것을 의도대로 매번 달라지는 값이 들어가게 하려면 어떻게 해야 할까요?

팩토리얼 팩토리얼을 구하는 것은 프로그래밍 연습에서 많이들 해보는 예제입니다. 다음과 같이 재귀적으로 정의되는 함수

```
def fact(n):
    if n<=0:
        return 1
    return n * fact(n-1)
```

를 expl3로도 물론 정의는 할 수 있는데요,

```
\ExplSyntaxOn
\NewDocumentCommand \rfacto { m }
{
    \factorial_rec_fn:n { #1 }
}

\cs_new:Npn \factorial_rec_fn:n #1
{
    \int_compare:nTF { #1 <= 1 }
    {
        1
    }
    {
        \int_eval:n { #1 * \factorial_rec_fn:n { \int_eval:n { #1 - 1 } } }
    }
}

\ExplSyntaxOff

\rfacto{10}
```

3628800

이미 설명한 대로, 직접 값을 돌려주는 위의 정의보다는 다음처럼 하는 것이 대체로 더 안전합니다.

```
\ExplSyntaxOn
\NewDocumentCommand \factorialp { m }
{
    \int_set:Nn \g_tmpa_int { \c_one_int }
    \facto_func:n { #1 }
    \int_use:N \g_tmpa_int
}

\cs_new:Npn \facto_func:n #1
{
    \int_compare:nT { #1 > 1 }
    {
        \int_gset:Nn \g_tmpa_int { \g_tmpa_int * #1 }
        \facto_func:n { \int_eval:n { #1 - 1 } }
    }
}

\ExplSyntaxOff
\factorialp{10}
```

3628800

expl3의 interface function들인 `\clist_item:Nn`이나 `\int_set:Nn`, `\int_add:Nn` 같은 데서 n 부분에 굳이 `\int_eval:n`을 쓰지 않아도 됐는데, 위의 사용자 정의 함수 `\facto_func:n`은 왜 이걸 인자에다가 써주어야 하는지에 대해 앞서 설명하였습니다. **사용자 정의 함수에서 인자의 확장에 대한 책임은 작성자에게 달려 있다**는 것입니다.

아무튼 팩토리얼에 대하여 재귀 함수 연습을 위한 목적으로는 이런 걸 해볼 수 있지만 실용적으로 팩토리얼을 써야 할 때는 다음처럼 합니다.
fp 자료형의 함수 가운데 `fact`가 있습니다.

```
\ExplSyntaxOn
\fp_eval:n { fact ( 13 ) } \\
\fp_eval:n { fact ( 25 ) }
\ExplSyntaxOff
```

```
6227020800
15511210043330990000000000
```

큰 수 다루기는 다음 기회에 따로 공부할 기회가 있겠지만 미리 bnumexpr라는 xint 엔진을 사용하는 패키지 하나를 사용해보면

```
\thebnumexpr 13!\relax \\
\thebnumexpr 25!\relax
```

```
6227020800
15511210043330985984000000
```

이 두 결과에 차이가 나는 이유는 fp 자료형이 유효숫자 16자리의 근삿값 계산을 하기 때문입니다.

연습문제

기본 3. 30이하의 정수 인자를 받아서 2^n 연산의 결과를 출력하는 함수 `\bin_power:n`을 작성하여라. 단 fp 자료형의 power 연산자 `**`를 쓰지 말고 int로 계산하여야 하며, 그 결과는 `\fp_eval:n { 2**#1 }`과 같아야 한다.

다음 관계식을 이용하겠습니다.

$$2^n = 2 \times 2^{n-1} \quad (2^0 = 1,\ 1 \le n \le 30)$$

```
\ExplSyntaxOn
\cs_new:Npn \bin_power:n #1
{
    \int_compare:nTF { #1 > 30 }
    {
        number~too~large \par
    }
    {
        \int_case:nnF { #1 }
        {
            { 0 } { 1 }
            { 1 } { 2 }
        }
        {
            \int_eval:n { 2 * \bin_power:n { #1 - 1 } }
        }
    }
}

\bin_power:n { 10 }

\ExplSyntaxOff
```

1024

연습문제

실력 4. 인자로 2진수가 주어진다. 이를 10진수로 바꾸어서 출력하는 함수를, expl3가 제공하는 진법 변환 함수를 차용하지 않고 작성하여라. 필요하다면 연습문제 3에서 작성한 `\bin_power:n`을 활용하여라.

□□□군의 코드는 다음과 같습니다.

```
\tl_set:Nn \l_tmpa_tl { #1 }
\tl_reverse:N \l_tmpa_tl

\int_zero:N \l_tmpa_int
\int_set:Nn \l_tmpb_int { 1 }
\tl_map_inline:Nn \l_tmpa_tl
  {
    \int_add:Nn \l_tmpa_int { \l_tmpb_int * ##1 }
    \exp_args:NNo \int_set:Nn \l_tmpb_int { \l_tmpb_int * 2 }
  }
\int_use:N \l_tmpa_int
```

문자열을 뒤집어서 처리했네요. 명시적으로 `\bin_power:n`을 사용하지는 않았지만 사실상 같은 방법을 사용하였습니다.
□□□군의 코드는 다음과 같습니다.

```
\int_zero:N \l_tmpa_int
\tl_map_inline:nn { #1 }
{
    \int_set:Nn \l_tmpa_int { \int_eval:n { \l_tmpa_int * 2 + ##1 } }
}
\int_use:N \l_tmpa_int
```

문자열을 뒤집지 않고도 2를 곱하여 한 자리씩 왼쪽으로 보낼 수 있음을 보여주었습니다.

만약 `\bin_power:n`을 반드시 사용하라는 조건이 붙었다면 대략 다음과 같이 하는 것도 가능합니다만 2진법의 10진법 전환이라는 목적에 비추어보면 계산이 효율적이지는 못합니다. 다만 외부 함수를 부르는 것, case를 쓰는 것 등을 보이는 의미에서 다음 코드도 붙여두기로 합니다.

```
\ExplSyntaxOn
\int_new:N \l_sum_int

\NewDocumentCommand \mytodec { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_reverse:N \l_tmpa_tl

    \int_zero:N \l_tmpa_int
    \int_zero:N \l_sum_int

    \tl_map_inline:Nn \l_tmpa_tl
    {
        \str_case:nn { ##1 }
        {
            { 0 } { }
            { 1 } {
                \int_add:Nn \l_sum_int { \bin_power:n { \l_tmpa_int } }
```

```
            }
        }

        \int_incr:N \l_tmpa_int
    }

    \int_use:N \l_sum_int
}

\mytodec{101111}

\ExplSyntaxOff
```

```
47
```

입력을 문자열로 보고 이를 뒤집어서 처리하고, 게다가 0과 1을 비교하는 데서 `\int_case:nn`가 아니라 `\str_case:nn`을 쓰고 있습니다. `\int_...` 함수가 항상 `\int_eval:n`하는 단계를 거쳐야 하기 때문에 살짝 부하가 걸린다는 것을 의식한 트릭이지만 사실은 큰 차이 있을까 싶습니다.
주의할 것은 첫 번째 자리의 지수가 0이라는 것입니다. 그래서 `\int_incr:N`을 inline 함수 맨 마지막에 두었습니다.

진법변환 expl3의 interface 함수로 진법변환에 쓰는 것은 다음과 같습니다.

- `\int_to_bin:n`, `\int_from_bin:n`
- `\int_to_hex:n`, `\int_from_hex:n`
- `\int_to_oct:n`, `\int_from_oct:n`
- `\int_to_base:nn`, `\int_from_base:nn`

이름이 모든 것을 말해주므로 설명이 필요없을 듯하고요. 임의의 진법으로의 변환에 쓰이는 `\int_to_base:nn` 정도를 주의해서 보면 될 듯합니다. `\int_to_hex`와 `\int_to_base`는 뒤의 첫 글자를 대문자로 쓴 `\int_to_Hex:n` 또는 `\int_to_Base:nn`라는 변형이 있는데 수 대신 사용하는 알파벳 문자를 대문자로 쓰게 해줍니다.

```
\ExplSyntaxOn
\int_to_base:nn { 10 } { 7 } \\
\int_to_Base:nn { 300 } { 32 }
\ExplSyntaxOff
```

```
13
9C
```

원래 문제에 대하여 이 함수를 쓰면

```
\ExplSyntaxOn
\int_from_bin:n { 101110 }
\ExplSyntaxOff
```

```
46
```

연습문제

기본 1. 주어지는 자연수를 우리말로 읽어서 그 결과를 한글로 출력하여라. 단 입력되는 숫자는 5자리 이하로 한다.

각 자리에 0이 오는 경우에는 단위까지 포함하여 읽지 않습니다. 1인 경우, 이것이 1의 자리일 때는 "일"을 새기고, 그 윗자리(만까지)에서는 단위만 읽습니다.

```
\ExplSyntaxOn
\clist_const:Nn \c_d_clist { {}, 이, 삼, 사, 오, 육, 칠, 팔, 구 }
\clist_const:Nn \c_u_clist { {}, 십, 백, 천, 만 }

\tl_new:N \l_res_tl

\NewDocumentCommand \numtohangul { m }
{
    \tl_clear:N \l_res_tl
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_reverse:N \l_tmpa_tl

    \int_zero:N \l_tmpa_int

    \tl_map_inline:Nn \l_tmpa_tl
    {
        \int_incr:N \l_tmpa_int
        \int_compare:nF { ##1 == 0 }
        {
            \bool_if:nT
            { \int_compare_p:n { ##1 == 1 } && \int_compare_p:n { \l_tmpa_int
             ↪  == 1 } }
            {
                \tl_put_left:Nn \l_res_tl { 일 }
            }

            \tl_put_left:Nx \l_res_tl { \clist_item:Nn \c_u_clist {
             ↪  \l_tmpa_int } }
            \tl_put_left:Nx \l_res_tl { \clist_item:Nn \c_d_clist { ##1 } }
        }
    }

    \l_res_tl
}

\ExplSyntaxOff

\numtohangul{10211}\quad \numtohangul{1002}
```

만이백십일　천이

이번에도 문자열을 뒤집어서 처리했는데요, 그 대신 `\tl_put_left:Nn`를 이용하여 역순으로 읽은 결과를 쌓아간 것입니다.
□□□군이 만 단위 띄어쓰기를 구현했는데요, 여기에 적용하려면 `\c_u_clist`의 마지막 아이템을

```
\clist_const:Nn \c_u_clist { {}, 십, 백, 천, 만~ }
```

로 하면 될 듯하네요. 그런데 이렇게 하면 `\numtohangul{10000}`의 경우에 마지막에 스페이스 하나가 붙는 위험이 있습니다. 안전하게 하기 위해서

```
\regex_replace_once:nnN { \s $ } { } \l_res_tl
```

을 마지막에 실행해주면 끝의 스페이스를 제거할 수 있습니다.
이 아이디어는 곧바로 6자리 이상의 수에 대해서도 확장가능합니다. 조금 복잡해지기는 하겠지만. 예를 들면 억 이후로는 100,000,000을 "일억"이라고 읽어서 "일"을 새겨주는 것을 주의한다든가.....

숙제에 대한 코멘트

"일, 십, 백, 천, 만"의 단위와 "일, 이, 삼, ..., 구"의 자릿수를 "리스트"에 넣고 이를 `item` 함수로 참조하여 문자열을 구성한다는 아이디어를 모두 구현하였습니다.
실제 처리에서는 조금씩 차이를 보이는데, □□□군은 tl 리스트를, □□□군은 clist를 사용하였습니다.
입력되는 숫자를 처리하는 순서는 inline mapping 시의 카운터를 역순으로 세는 방법을 적용하였네요.
`\int_decr:N`이나 −1 하는 방식으로 세었어요.
0과 1을 처리하는 것은 구현은 조금 다르지만 원하는 목적을 잘 달성한 것으로 보입니다.
□□□군의 코드는 다음과 같습니다.

```
\ExplSyntaxOn
\tl_gset:Nn \g_prefix_tl { 일이삼사오육칠팔구 }
\tl_gset:Nn \g_suffix_tl { 십백천만 }

\NewDocumentCommand \numtohangul { m }
  {
    \int_set:Nn \l_tmpa_int { \tl_count:n { #1 } - 1 }
    \tl_map_inline:nn { #1 }
      {
        \bool_if:nT
          {
            \int_compare_p:n { \l_tmpa_int > 3 } ||
            \int_compare_p:n { \l_tmpa_int == 0 } ||
            \int_compare_p:n { ##1 != 1 }
          }
          {
            \tl_item:Nn \g_prefix_tl { ##1 }
          }
        \int_compare:nT { ##1 != 0 }
          {
            \tl_item:Nn \g_suffix_tl { \l_tmpa_int }
          }
        \int_compare:nT { \l_tmpa_int == 4 } { ~ }
        \int_decr:N \l_tmpa_int
      }
  }
\ExplSyntaxOff

\numtohangul{2019}\\
\numtohangul{11019}\\
\numtohangul{909}
```

이천십구
일만 천십구
구백구

□□□군의 코드는 다음과 같습니다.

```
\ExplSyntaxOn
\clist_new:N \l_digit_clist
```

```
\clist_set:Nn \l_digit_clist { 일, 십, 백, 천, 만 }
\clist_new:N \l_num_clist
\clist_set:Nn \l_num_clist { 일, 이, 삼, 사, 오, 육, 칠, 팔, 구 }
\NewDocumentCommand \numtohangul { m }
{
    \int_set:Nn \l_tmpa_int { \tl_count:n { #1 } }
    \tl_map_inline:nn { #1 }
    {
        \int_case:nnTF { ##1 }
        {
            { 0 } {  }
            { 1 } { \clist_item:Nn \l_digit_clist { \l_tmpa_int } }
        }
        { }
        {
            \clist_item:Nn \l_num_clist { ##1 }
            \int_compare:nTF { \l_tmpa_int == 1 }
            { }
            {
                \clist_item:Nn \l_digit_clist { \l_tmpa_int }
            }
        }
        \int_add:Nn \l_tmpa_int { -1 }
    }
}
\ExplSyntaxOff

\numtohangul{102}
\numtohangul{120}
\numtohangul{123}
\numtohangul{2019}
\numtohangul{98765}
\numtohangul{101}
```

백이 백이십 백이십삼 이천십구 구만팔천칠백육십오 백일

연습문제

실력 2. 두 수를 인자로 받아 그 최대공약수를 출력하는 명령 \mygcd를 작성하여라.

```
def mygcd(a,b):
    if a<b:
        a,b=b,a
    while b != 0:
        t = a%b
        a,b=b,t
    return a
```

위의 알고리즘을 그대로 expl3로 옮겨쓰면 다음과 같이 됩니다.

```
\ExplSyntaxOn

\int_new:N \l_tmpc_int

\NewDocumentCommand \mygcd { m m }
{
    \int_set:Nn \l_tmpa_int { \int_max:nn { #1 } { #2 } }
    \int_set:Nn \l_tmpb_int { \int_min:nn { #1 } { #2 } }

    \int_while_do:nn { \l_tmpb_int != 0 }
    {
        \int_set:Nn \l_tmpc_int
            { \int_mod:nn { \l_tmpa_int } { \l_tmpb_int } }
        \int_set_eq:NN \l_tmpa_int \l_tmpb_int
        \int_set_eq:NN \l_tmpb_int \l_tmpc_int
    }

    \int_use:N \l_tmpa_int
}

\ExplSyntaxOff
\mygcd{16}{24}
```

```
8
```

예를 들어,

```
a,b=b,a-b
```

이런 python 코드를 expl3로 쓴다고 할 때,

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int { 10 }
\int_set:Nn \l_tmpb_int { 4 }

\int_set_eq:NN \l_tmpa_int \l_tmpb_int
\int_set:Nn \l_tmpb_int { \l_tmpa_int - \l_tmpb_int }

\int_use:N \l_tmpa_int; ~ \int_use:N \l_tmpb_int
\ExplSyntaxOff
```

```
4; 0
```

보다시피 결과가 잘못되어 있습니다. 그 이유는 `\l_tmpa_int` 값이 변화한 이후에 이것을 계산에 사용했기 때문이므로, 반드시 temp를 하나 써서

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int { 10 }
\int_set:Nn \l_tmpb_int { 4 }

\int_set:Nn \l_tmpc_int { \l_tmpa_int - \l_tmpb_int }

\int_set_eq:NN \l_tmpa_int \l_tmpb_int
\int_set_eq:NN \l_tmpb_int \l_tmpc_int

\int_use:N \l_tmpa_int; ~ \int_use:N \l_tmpb_int
\ExplSyntaxOff
```

```
4; 6
```

이와 같이 하여야 합니다. python의 문법이 너무 간단해서 생기는 문제이고 옛날 언어들에서는 다 이렇게 했지요.

숙제에 대해서는 따로 할 말이 없습니다. python 코드를 충실히 잘 번역할 수 있었던 것으로 봅니다.

연습문제

기본 1. 다음 도표의 빈 칸을 채워 완성하여라.

	pt	mm	pc	in
pt	1.00000	0.35146		
mm	2.84526	1.00000		
pc			1.00000	
in				1.00000

`\dim_to_decimal_in_unit:nn`이면 다 될 것입니다. 문제는 얼마나 간단하게 표를 작성할 것이냐겠죠.

제1단계: 그냥 써넣기 이것저것 고민하지 말고 그냥 각 셀을 위의 명령으로 채우는 것입니다.

```
\ExplSyntaxOn
\cs_new:Npn \cnv_cnv:nn #1 #2 { \dim_to_decimal_in_unit:nn { 1 #1 } { 1 #2 }
 ↪  }
\cs_set_eq:NN \cnvcnv \cnv_cnv:nn
\begin{tabular}{c|c|c|c|c}
   & pt & mm & pc & in \\ \hline
pt & \cnvcnv{pt}{pt} & \cnvcnv{pt}{mm} & \cnvcnv{pt}{pc} & \cnvcnv{pt}{in} \\
 ↪  \hline
mm & \cnvcnv{mm}{pt} & \cnvcnv{mm}{mm} & \cnvcnv{mm}{pc} & \cnvcnv{mm}{in} \\
 ↪  \hline
pc & \cnvcnv{pc}{pt} & \cnvcnv{pc}{mm} & \cnvcnv{pc}{pc} & \cnvcnv{pc}{in} \\
 ↪  \hline
in & \cnvcnv{in}{pt} & \cnvcnv{in}{mm} & \cnvcnv{in}{pc} & \cnvcnv{in}{in} \\
 ↪  \hline
\end{tabular}
\ExplSyntaxOff
```

	pt	mm	pc	in
pt	1	0.35146	0.08333	0.01384
mm	2.84526	1	0.2371	0.03937
pc	12	4.21754	1	0.16605
in	72.26999	25.40013	6.0225	1

그냥은 `\dim_to_decimal_in_unit:nn`이라는 게 너무 길어서 단축 명령을 정의했습니다. 매번 `{1mm}`처럼 1을 써야하는 것도 귀찮아서 생략할 수 있게 했습니다.

제2단계: 반복줄이기 실제 입력해 보면 이것은 "똑같은 명령을 너무 많이 반복"하고 있습니다. 이런 건 어떻게든 해봐야겠죠. `{pt,mm,pc,in}`이 차례로 들어 있는 clist를 상정하면 위의 코드는 이를테면,

```
for i in range(1,4):
    for j in range(1,4):
        cnvcnv(i,j)
        if j < 4:
            "&"
    else:
            "\\ \hline"
```

이런 거니까, 다음처럼 해보겠습니다.

```
\ExplSyntaxOn
\protected\def\newlinehline { \tabularnewline \hline }
\cs_set_eq:NN \cnvcnv \cnv_cnv:nn

\clist_set:Nn \l_tmpa_clist { pt, mm, pc, in }

\tl_set:Nn \l_tablines_tl { & pt & mm & pc & in \newlinehline }

\clist_map_inline:Nn \l_tmpa_clist
{
    \tl_put_right:Nn \l_tablines_tl { #1 & }

    \int_zero:N \l_tmpa_int
    \clist_map_inline:Nn \l_tmpa_clist
    {
        \tl_put_right:Nx \l_tablines_tl { \cnvcnv{#1}{##1} }
        \int_incr:N \l_tmpa_int
        \int_compare:nTF { \l_tmpa_int < \clist_count:N \l_tmpa_clist }
        {
            \tl_put_right:Nn \l_tablines_tl { & }
        }
        {
            \tl_put_right:Nn \l_tablines_tl { \newlinehline }
        }
    }
}

\begin{tabular}{c|c|c|c|c}
\l_tablines_tl
\end{tabular}
\ExplSyntaxOff
```

	pt	mm	pc	in
pt	1	0.35146	0.08333	0.01384
mm	2.84526	1	0.2371	0.03937
pc	12	4.21754	1	0.16605
in	72.26999	25.40013	6.0225	1

쓸데없는 오류를 줄이기 위해서 `\\ \hline`을 하나의 protected macro로 묶었습니다. 이 방법은 tabular를 그릴 때 흔히 사용하는 것이므로 잘 보아두어야 합니다. 그리고 tabular 환경 안에 올 것들을 `\l_tablines_tl`이라는 변수에 모두 다 저장한 후에 tabular 환경 안에서 불렀습니다. 물론 이미 배운 대로 `&`를 `\c_alignment_token`이라고 써도 좋습니다.

위의 코드에서 주의할 점 하나를 지적합니다. 16행에 보면 `\cnvcnv{#1}{##1}`이 있습니다. 지금 clist map이 안과 밖 두 겹으로 돌고 있고 안쪽 map이 넘겨주는 current item (j)은 `##1`, 바깥쪽 map이 넘겨주는 item (i)은 `#1`인데요, 만약 이 코드가 다른 함수의 정의 안에 들어간다면 어떻게 될까요? 그러면 `#1`은 함수 자체의 인자를 가리킬 것이므로 map의 item을 가리키기 위해서 `##1`과 `####1`이 되어야 하는데 `#`의 중첩은 두 개까지만 허용하는 것으로 보기로 하였으므로, 다른 함수의 정의 안에서 이 코드를 쓴다면 중첩된 두 clist map 문 가운데 하나 이상을 `\clist_map_function:NN`으로 처리해야 마땅합니다.

제3단계: 일반화 이번에는 아예 저 list를 인자로 주어서 표를 그리도록 하는데 인자를 원하는 대로 줄 수 있도록 해보고 싶습니다. `\dimtable`이라는 명령을 하나 만들고

```
\dimtable{pt,mm,in,cc,dd}
```

이렇게 주면 6 × 6 tabular를 그려주게 하자는 거지요. (참고로, `sp`는 숫자가 너무 커지기 때문에 arithmetic overflow를 보일 수 있습니다. 이 단위는 TEX이 내부적으로 사용하도록 두고 꼭 값을 알고 싶다면 pt에 65536

을 곱해보도록 하세요.)
이 때 어려운 점은 tabular의 column spec 인자를 구성하는 것입니다. 그려야 할 tabular의 컬럼이 3개라면

```
\begin{tabular}{c|c|c}
```

이렇지만 만약 5개라면

```
\begin{tabular}{c|c|c|c|c}
```

이렇게 되어야 할 것이기 때문입니다.
이 문제의 해결책은 tabular column spec의 *(asterisk) notation입니다.

```
\begin{tabular}{*5{c}}
```

이것이

```
\begin{tabular}{ccccc}
```

와 같다는 것을 이용합니다.
`\dimtable{pt,sp,cc,in}`과 같이 들어온다면, 이것을 clist에 넣었을 때 `\clist_count:N #1`의 값이 4일 테니까,

```
\begin{tabular}{ c *\clist_count:N #1 { |c } }
```

이렇게 해주면

```
\begin{tabular}{c|c|c|c|c}
```

이게 된다는 의미입니다.
다만 tabular의 column spec 인자는 무조건 제일 먼저 확장되는 부분이라서 무슨 일이 있을지 모르니까 안전하게 하기 위해서 `\clist_count:N #1`과 같은 부분을 직접 쓰지 말고 이 값을 미리 구한 다음 tl로 전달하는 것이 좋겠습니다. 이 tl은 tmpa같은 걸 쓰지 말고 명시적인 이름을 붙여두기로 합니다.

```
\tl_set:Nx \l_colspec_tl { \clist_count:N #1 }
\begin{tabular} { c *\l_colspec_tl { |c } }
```

clist보다 seq가 좋다고 하고 어차피 카운터 체크를 해서 마지막 컬럼인지 아닌지에 따라 `&`이나 `\\ \hline`을 구분해야 하니까 `\seq_indexed_map_...`을 써볼 수 있을 것입니다.

```
\ExplSyntaxOn
\protected\def\newlinehline { \tabularnewline \hline }
\cs_set_eq:NN \cnvcnv \cnv_cnv:nn

\NewDocumentCommand \dimtable { m }
{
    \seq_set_from_clist:Nn \l_tmpa_seq { #1 }
%%% col spec tl
    \int_set:Nn \l_tmpa_int { \seq_count:N \l_tmpa_seq }
    \tl_set:Nx \l_colspec_tl { c * \int_use:N \l_tmpa_int { |c } }

%%% head line
    \tl_set:Nx \l_tablines_tl
    {
        & \seq_use:Nn \l_tmpa_seq { & } \newlinehline
    }

%%% build up tablines
    \seq_map_function:NN \l_tmpa_seq \build_tablines:n
```

```
%%% print out tabular
    \print_tabular:NN \l_colspec_tl \l_tablines_tl
}

\cs_new:Npn \build_tablines:n #1
{
    \tl_put_right:Nn \l_tablines_tl { #1 & }

    \seq_indexed_map_inline:Nn \l_tmpa_seq
    {
        \tl_put_right:Nx \l_tablines_tl { \cnvcnv { #1 } { ##2 } }
        \int_compare:nTF { ##1 < \l_tmpa_int } %%% \l_tmpa_int == seq count
        {
            \tl_put_right:Nn \l_tablines_tl { & }
        }
        {
            \tl_put_right:Nn \l_tablines_tl { \newlinehline }
        }
    }
}

\cs_new:Npn \print_tabular:NN #1 #2
%%% #1 = colspec, #2 = tablines
{
    \exp_args:Nnx
    \begin{tabular}{#1}
      #2
    \end{tabular}
}

\ExplSyntaxOff

\dimtable{pt,in,mm,cc,dd}
```

	pt	in	mm	cc	dd
pt	1	0.01384	0.35146	0.07788	0.93457
in	72.26999	1	25.40013	5.62846	67.54158
mm	2.84526	0.03937	1	0.22159	2.6591
cc	12.8401	0.17767	4.5128	1	12.00002
dd	1.07	0.0148	0.37607	0.08333	1

45행의 `\exp_args:Nnx`를 주의하여야 합니다. 보조 함수 `\print_tabular:NN`의 첫 인자는 tl macro 입니다. 그러므로 이것을 확장해주지 않으면 tabular가 불만 표시를 할 게 틀림없습니다. 이 문제를 피해가는 다른 방법으로 `\exp_args:Nnx`를 적지 않는다면 이 함수의 인자형을 `:NN`이 아니라 `:nN`으로 정의한 후에

```
\cs_generate_variant:Nn \print_tabular:nN { VN }
```

을 선언해두고 `\dimtable`의 마지막 이 함수를 부를 적에

```
\print_tabular:VN \l_colspec_tl \l_tablines_tl
```

으로 할 수도 있습니다. 어떤 경우든 colspec은 평이한 문자열로 전달되어야 한다는 것입니다.

마지막 tuning 이런 식으로 소수점 있는 숫자가 많은 표는 소수점 기준으로 정렬해서 보여주면 좋습니다. tabular에서 소수점 기준 정렬은 완전히 다른 종류의 문제이고 이를 해결하는 패키지들이 몇 있습니다만 여기서는 단순히 소수점 아래 자릿수를 고정시켜놓고 우측정렬하는 방식으로 해보려고 합니다.
일단 모든 숫자를 전부 소수점 아래 세 자리만 보여주게 하려면
colspec을 `c *\l_colspec_tl { |c }`가 아니라 (10행)

```
\tl_set:Nx \l_colspec_tl { c * \int_use:N \l_tmpa_int { |r } }
```

으로 합니다. 첫 컬럼을 제외하고는 모두 우측정렬입니다.
그리고 27행의 tablines에 넣는 과정에서 우리가 만들어둔 `\esgfpformat` 명령을 써보기로 합니다.

```
\tl_put_right:Nx \l_tablines_tl
{
    \esgfpformat [3] { \cnvcnv { #1 } { ##2 } }
}
```

이것만으로 다음과 같은 결과가 될 것입니다.

```
\dimtable{mm,cm,pt}
```

	mm	cm	pt
mm	1.000	0.100	2.845
cm	10.000	1.000	28.453
pt	0.351	0.035	1.000

만약 1일 때는 그 뒤에 꼬리 0를 붙이고 싶지 않다면
숫자를 tablines tl에 넣는 부분을 다음과 같이 수정합니다.

```
\tl_put_right:Nx \l_tablines_tl
{
    \fp_compare:nTF { \cnvcnv { #1 } { ##2 } == 1 }
    {
        1
    }
    {
        \esgfpformat [3] { \cnvcnv { #1 } { ##2 } }
    }
}
```

```
\dimtable{mm,cm,pt,in}
```

	mm	cm	pt	in
mm	1	0.100	2.845	0.039
cm	10.000	1	28.453	0.394
pt	0.351	0.035	1	0.014
in	25.400	2.540	72.270	1

`\esgfpformat`에 대하여는 이미 공부한 적이 있으므로 여기서 설명하지 않겠습니다.
마지막 남은 문제는 제일 첫 행만은 우측 정렬이 아니라 가운데 오면 좋겠다는 것 정도겠죠. 이를 위해서는 tabular를 다음과 같이 만들어야 합니다. head line이라는 주석이 붙어 있는 12–16행 부분을

```
%%% head line
    \tl_set:Nx \l_tablines_tl
    {
        &
    }
    \seq_indexed_map_inline:Nn \l_tmpa_seq
```

```
    {
        \int_compare:nTF { ##1 < \seq_count:N \l_tmpa_seq }
        {
            \tl_put_right:Nn \l_tablines_tl
                { \multicolumn{1}{c|}{ ##2 } }
            \tl_put_right:Nn \l_tablines_tl { & }
        }
        {
            \tl_put_right:Nn \l_tablines_tl
                { \multicolumn{1}{c}{ ##2 } }
            \tl_put_right:Nn \l_tablines_tl { \newlinehline }
        }
    }
```

이렇게 수정합니다. 이것은 매 셀에 \multicolumn을 주어서 강제로 c 정렬을 시킨 것인데, 이런 일을 좀 쉽게 하게 해주는 makecell 패키지를 써도 좋습니다. \multicolumn 명령과 관련된 설명은 다 아는 것으로 보아서 더 하지 않겠습니다.
결과는 이러네요.

```
\dimtable{pt,mm,cm,in}
```

	pt	mm	cm	in
pt	1	0.351	0.035	0.014
mm	2.845	1	0.100	0.039
cm	28.453	10.000	1	0.394
in	72.270	25.400	2.540	1

이 모양을 만들어보라는 것이 원래 출제 의도였습니다. 간단한 문제지만 제법 많은 걸 해본 거 같네요.

숙제에 대한 코멘트 □□□군이 다음과 같은 코드를 보내왔는데요, 간결하고 좋은 해결책입니다.

```
\ExplSyntaxOn

\cs_new:Npn \convert_from:n #1
  {
    \dim_to_decimal_in_unit:nn { #1 } { 1pt } &
    \dim_to_decimal_in_unit:nn { #1 } { 1mm } &
    \dim_to_decimal_in_unit:nn { #1 } { 1pc } &
    \dim_to_decimal_in_unit:nn { #1 } { 1in } \\ \hline
  }

\begin{center}
\begin{tabular}{c|c|c|c|c}
     & pt & mm & pc & in \\ \hline
  pt & \convert_from:n { 1pt }
  mm & \convert_from:n { 1mm }
  pc & \convert_from:n { 1pc }
  in & \convert_from:n { 1in }
\end{tabular}
\end{center}

\ExplSyntaxOff
```

	pt	mm	pc	in
pt	1	0.35146	0.08333	0.01384
mm	2.84526	1	0.2371	0.03937
pc	12	4.21754	1	0.16605
in	72.26999	25.40013	6.0225	1

연습문제

기본 2. LaTeX에 \settowidth와 \settoheight, \settodepth라는 명령이 있다. 이의 사용법은 다음과 같다.

```
\newlength{\mylen}
\settowidth{\mylen}{beautiful}
\the\mylen
```

39.75pt

이것과 동일한 작용을 하는 \SettoWidth, \SettoHeight, \SettoDepth를 expl3로 만들고 \SettoTotalheight도 작성해보아라.

\SettoWidth를 예로 들어보겠습니다. 이건 그냥 #1으로 들어오는 매크로 이름에다가 #2를 넣은 박스의 width를 넘겨주면 간단히 끝날 테지요.

```
\ExplSyntaxOn
\NewDocumentCommand \SettoWidth { m m }
{
    \hbox_set:Nn \l_tmpa_box { #2 }
    \dim_set:Nn #1 { \box_wd:N \l_tmpa_box }
}
\ExplSyntaxOff
%%% ---- test ----
\newlength\tmplen
\SettoWidth{\tmplen}{beautiful}
\the\tmplen
```

39.75pt

만약 #1의 길이 변수가 정의되어 있지 않다면 어떻게 하지요? 위와 같이 이 명령에서 아무런 처리를 하지 않으면 5행을 실행할 때 LaTeX이 그런 매크로는 정의되지 않았다고 에러를 토하면서 정지합니다. 그게 싫으면 에러 처리 루틴을 두어도 좋습니다.
다른 것도 이렇게 하면 되기는 하는데....., 똑같은 코드를 반복해서 쓴다는 게 좀 맘에 걸립니다. 그럴 때는 다음과 같이 할 수 있습니다.

```
\ExplSyntaxOn
\cs_new:Npn \set_to_dim:nnn #1 #2 #3
{
    \hbox_set:Nn \l_tmpa_box { #3 }
    \dim_set:Nn #1 { \use:c { box_ #2 :N } \l_tmpa_box }
}

\prop_set_from_keyval:Nn \l_tmpa_prop { Width = wd, Height = ht, Depth = dp }

\prop_map_inline:Nn \l_tmpa_prop
{
    \cs_set_protected_nopar:cpn { Setto #1 } ##1 ##2
    {
        \set_to_dim:nnn ##1 { #2 } { ##2 }
    }
}

\NewDocumentCommand \SettoTotalheight { m m }
{
```

```
    \hbox_set:Nn \l_tmpa_box { #2 }
    \dim_if_exist:NTF #1
    {
        \dim_set:Nn #1 { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
    }
    {
        dimension~#1~is~not~defined
    }
}
\ExplSyntaxOff

%%%% ---- test ----
\newlength\testlen
\SettoWidth{\testlen}{beautiful}
\the\testlen \quad
\SettoHeight{\testlen}{beautiful}
\the\testlen \quad
\SettoDepth{\testlen}{beautiful}
\the\testlen \quad
\SettoTotalheight{\testlen}{beautiful}
\ExplSyntaxOn
\esg_fp_format:nn { 2 } { \dim_to_decimal:n { \testlen } } pt
\ExplSyntaxOff
```

39.75pt 7.28pt 0.19998pt 7.48pt

line 8:
prop에 `Width=wd`와 같이 넣었는데, 앞의 Width는 `\Setto`<거시기>할 적의 <거시기>에 해당하는 단어이고, wd는 `\box_`<거시기>`:N`할 적의 <거시기>에 해당하는 단어입니다. 즉 `\SettoWidth`라는 명령을 정의하면 그 명령을 정의하는 명령에서는 `\box_wd:N`을 쓰라고 할 참입니다.
line 12:
`\cs_set_protected_nopar:Npn`을 썼는데요, 끝의 `_nopar`는 이 명령이 어떤 경우에도 `\par`를 포함하지 않을 함수라는 의미입니다. LaTeX의 `\newcommand*`에 해당합니다. `_protected`는 ε-TeX의 `\protected`와 비슷하게 해당 매크로를 protect해주는 것입니다.
실제 이렇게 함수를 정의하면 `\NewDocumentCommand`라고 한 것과 유사한 효과를 가집니다. `\cs_new`와 달리 `\cs_set`은 정의하는 함수 이름에 `:`(콜론)이 없어도 일단 통과는 시켜준다는 것을 배웠습니다.
이것을 쓴 이유는 c형 인자를 쓰기 위해서입니다. 즉 Width가 들어오면 `\SettoWidth`를, Height가 들어오면 `\SettoHeight`를 정의하라는 의미인 것입니다.
여기에 굳이, 꼭, 반드시 `\NewDocumentCommand`를 써야 한다면 어떻게 할까요? 다음과 같이 합니다.

```
\exp_after:wN \NewDocumentCommand \use:c { Setto #1 } { m m }
```

자세한 설명은 생략합니다만 여기서 `\NewDocumentCommand \use:c`라고, 뒤의 것의 확장 없이 바로 쓸 수 없음을 지적해둡니다. 이유는 이렇게 쓰면 `\use:c`라는 매크로를 새로 정의하려고 한다는 의미가 되어서 에러가 나올 수밖에 없기 때문입니다. 이 코드의 `\exp_after:wN`은 사실상 `\expandafter`를 expl3 식으로 쓴 것에 불과합니다. 당연히 `\expandafter`로 해도 상관없습니다.
마지막으로 이 대목에서 `##1`, `##2`라고 #이 두 번씩 쓰인 것은 이해할 수 있겠지요? 이 블럭은 `\prop_map_...` 블럭 안이기 때문에 여기에서 #1과 #2란, 이번 mapping 순번에서의 key와 value를 의미하지 `\cs_set`으로 정의되는 함수의 parameter가 될 수 없는 것입니다.
line 2:
명령을 정의하는 명령 `\set_to_dim:nnn`은 세 개의 인자를 갖습니다. #1은 길이를 저장할 매크로, #2는 길이의 종류(wd, ht, dp), 그리고 #3은 길이를 잴 텍스트입니다.

14행에서 이 함수를 어떻게 불렀는지 잘 보세요. ##1과 ##2는 무엇이고 #2는 무엇인지 헷갈리지 않도록 하세요.

연습문제

기본 3. 다음 문장에서, 각 단어의 (문장부호를 제외한) 길이를 측정하여 평균값을 구하고, 그 문단의 margin에 `\rule{<평균값>}{10pt}`의 막대를 그려라.

이 문제를 해결하면, 다음과 같은 출력을 얻습니다.

남방의 비는 차갑고 단단하고 찬란한 눈꽃으로 변하는 일이 절대 없다. 식자들은 남방의 비가 단조롭다고 생각하는데 비 자신도 그것을 불행으로 여기는지 어떤지, 강남의 눈은 그래도 촉촉하고 윤기가 돌아 아름답기 그지없다. 그것은 아직 눈뜨지 않은 청춘의 소식이며, 건강한 처녀의 살결이다.
눈 덮인 들에는 핏빛의 붉은 동백이며, 흰빛에 푸르름이 감도는 외겹 매화며, 경쇠 모양의 진노랑 새양꽃이 있고, 눈 밑에는 아직도 파랗게 언 잡초들이 있다. 나비는 분명 없었다. 꿀벌이 동백꽃과 매화꽃의 꿀을 따러 왔었는지 아닌지, 나는 확실히 기억할 수 없다. 단지 나의 눈앞에는 겨울꽃이 핀 눈 덮인 들판에 바쁘게 날아다니는 수많은 벌들이 보이는 듯, 시끄럽게 붕붕거리는 소리가 들리는 듯했다.

문단과 단어 한 개의 문단만을 대상으로 하는 명령을 작성하라는 것이 핵심이어요. 이 문단 전체를 인자로 제공받는 명령으로 만들어서, 단어별로 길이를 측정해서 모두 합하고, 단어 수로 나누어서 평균 길이를 얻어보겠습니다. 단어의 길이를 재는 데는 앞서 만든 `\SettoWidth`를 쓰기로 하지요.

```
\ExplSyntaxOn
\dim_new:N \l_avgwordlen_dim
\NewDocumentCommand \mycmd { m }
{
    \seq_set_split:Nnn \l_tmpa_seq { ~ } { #1 }
    \dim_zero:N \l_tmpa_dim
    \seq_map_inline:Nn \l_tmpa_seq
    {
        \SettoWidth{ \l_tmpb_dim } { ##1 }
        \dim_add:Nn \l_tmpa_dim { \l_tmpb_dim }
    }

    \dim_set:Nn \l_avgwordlen_dim { \l_tmpa_dim / \seq_count:N \l_tmpa_seq }

%%% print the average length
    \dim_use:N \l_avgwordlen_dim
}
\ExplSyntaxOff

\mycmd{남방의 비는 차갑고 단단하고 찬란한 눈꽃으로 변하는 일이 절대 없다. 식자들은 남방의 비가 단
↪ 조롭다고 생각하는데 비 자신도 그것을 불행으로 여기는지 어떤지, 강남의 눈은 그래도 촉촉하고 윤
↪ 기가 돌아 아름답기 그지없다. 그것은 아직 눈뜨지 않은 청춘의 소식이며, 건강한 처녀의 살결이
↪ 다.}
```

30.17159pt

단어들의 평균길이는 그럭저럭 계산하는 거 같은데, "문장부호를 제외하고"라는 조건이 충족되지 않았습니다. 이 문단에서는 일단 마침표와 쉼표밖에 없으니까 그 둘만 제거하여 같은 일을 해보겠습니다.

```
\ExplSyntaxOn
%\dim_new:N \l_avgwordlen_dim
\NewDocumentCommand \mycmd { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_remove_all:Nn \l_tmpa_tl { , }
```

```
    \tl_remove_all:Nn \l_tmpa_tl { . }

    \seq_set_split:NnV \l_tmpa_seq { ~ } \l_tmpa_tl
    \dim_zero:N \l_tmpa_dim
    \seq_map_inline:Nn \l_tmpa_seq
    {
        \SettoWidth{ \l_tmpb_dim } { ##1 }
        \dim_add:Nn \l_tmpa_dim { \l_tmpb_dim }
    }

    \dim_set:Nn \l_avgwordlen_dim { \l_tmpa_dim / \seq_count:N \l_tmpa_seq }

%%% print the average length
    \dim_use:N \l_avgwordlen_dim
}
\ExplSyntaxOff

\mycmd{남방의 비는 차갑고 단단하고 찬란한 눈꽃으로 변하는 일이 절대 없다. 식자들은 남방의 비가 단
↪ 조롭다고 생각하는데 비 자신도 그것을 불행으로 여기는지 어떤지, 강남의 눈은 그래도 촉촉하고 윤
↪ 기가 돌아 아름답기 그지없다. 그것은 아직 눈뜨지 않은 청춘의 소식이며, 건강한 처녀의 살결이
↪ 다.}
```

29.74265pt

\marginpar의 정체 \marginpar는 floating object입니다. 그리고 page를 기준으로 그 위치와 크기가 결정됩니다. (tcolorbox의 tcblisting 안에 넣어봤자 잘 안 되는 이유는 그 때문입니다.)
이제 해야 할 일은 앞서 계산한 \l_avgwordlen_dim을 가지고 막대를 그려야 하는 건데, 이것을 문단의 처음에 두고 그 뒤에 (인자로 입력받은 텍스트를 식자하면 될 것 같기도 합니다.
그래서,

```
\ExplSyntaxOn
%\dim_new:N \l_avgwordlen_dim
\NewDocumentCommand \mycmd { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_remove_all:Nn \l_tmpa_tl { , }
    \tl_remove_all:Nn \l_tmpa_tl { . }

    \seq_set_split:NnV \l_tmpa_seq { ~ } \l_tmpa_tl
    \dim_zero:N \l_tmpa_dim
    \seq_map_inline:Nn \l_tmpa_seq
    {
        \SettoWidth{ \l_tmpb_dim } { ##1 }
        \dim_add:Nn \l_tmpa_dim { \l_tmpb_dim }
    }

    \dim_set:Nn \l_avgwordlen_dim { \l_tmpa_dim / \seq_count:N \l_tmpa_seq }

%%% marginpar
    \marginpar{\rule{\l_avgwordlen_dim}{10pt}}
    #1
}
\ExplSyntaxOff
```

이렇게 해서 식자를 하면 다음과 같이 됩니다.

> 남방의 비는 차갑고 단단하고 찬란한 눈꽃으로 변하는 일이 절대 없다. 식자들은 남방의 비가 단조롭다고 생각하는데 비 자신도 그것을 불행으로 여기는지 어떤지, 강남의 눈은 그래도 촉촉하고 윤기가 돌아 아름답기 그지없다. 그것은 아직 눈뜨지 않은 청춘의 소식이며, 건강한 처녀의 살결이다.

(여기서부터 `\marginpar`가 명령 안에 들어가기 때문에 위의 예시를 위한 examplebelow 같은 환경 안에 넣을 수 없습니다.) 그나저나 막대의 위치가 원하는 것보다 한 줄 정도 위에 찍히고 있습니다. 문단 시작 위치와 비슷했으면 좋겠는데 말이죠. 이런 일이 일어나는 이류를 상세하게 설명하려면 *The TEXbook*을 읽어서 horizontal mode, vertical mode 따위에 대하여 이해하면 쉬운데, 우리는 여기서 그런 건 "이미 다 아는 이야기"로 전제하고 있기 때문에 간단히,

> 문단 시작 위치에 온 매크로는 hmode로 아직 진입하지 않았을 가능성이 있다.

는 사실에 기초하여, `\leavevmode`를 두기로 합니다. expl3 버전은 `\mode_leave_vertical:`입니다. 이름 그대로 "즉시 vmode를 떠나서 hmode로 들어가라"는 것입니다.

```
\ExplSyntaxOn
%\dim_new:N \l_avgwordlen_dim
\NewDocumentCommand \mycmd { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_remove_all:Nn \l_tmpa_tl { , }
    \tl_remove_all:Nn \l_tmpa_tl { . }

    \seq_set_split:NnV \l_tmpa_seq { ~ } \l_tmpa_tl
    \dim_zero:N \l_tmpa_dim
    \seq_map_inline:Nn \l_tmpa_seq
    {
        \SettoWidth{ \l_tmpb_dim } { ##1 }
        \dim_add:Nn \l_tmpa_dim { \l_tmpb_dim }
    }

    \dim_set:Nn \l_avgwordlen_dim { \l_tmpa_dim / \seq_count:N \l_tmpa_seq }

%%% marginpar
    \mode_vertical_leave:
    \marginpar{\rule{\l_avgwordlen_dim}{10pt}}
    #1
}
\ExplSyntaxOff
```

남방의 비는 차갑고 단단하고 찬란한 눈꽃으로 변하는 일이 절대 없다. 식자들은 남방의 비가 단조롭다고 생각하는데 비 자신도 그것을 불행으로 여기는지 어떤지, 강남의 눈은 그래도 촉촉하고 윤기가 돌아 아름답기 그지없다. 그것은 아직 눈뜨지 않은 청춘의 소식이며, 건강한 처녀의 살결이다.
눈 덮인 들에는 핏빛의 붉은 동백이며, 흰빛에 푸르름이 감도는 외겹 매화며, 경쇠 모양의 진노랑 새양꽃이 있고, 눈 밑에는 아직도 파랗게 언 잡초들이 있다. 나비는 분명 없었다. 꿀벌이 동백꽃과 매화꽃의 꿀을 따러 왔었는지 아닌지, 나는 확실히 기억할 수 없다. 단지 나의 눈앞에는 겨울꽃이 핀 눈 덮인 들판에 바쁘게 날아다니는 수많은 벌들이 보이는 듯, 시끄럽게 붕붕거리는 소리가 들리는 듯했다.

그래도 bar가 글자 대비 조금 위로 올라간 것으로 보인다면 그것은 baseline 때문입니다. 글자들은 baseline에 걸쳐 있는데 rule은 baseline 위로 그려져 있기 때문이지요. 이 자리에 바로 bar를 그려보면 abcxyz▮와 같이 baseline 위로 그려집니다. 원한다면 글자의 depth를 계산하여 그만큼 끌어내리면 됩니다만 여기서는

거기까지 하지는 않겠습니다. 또한 marginpar는 어차피 "떠다니는" 물건이라서 정확하게 원하는 위치에 오지 않는 경우도 많습니다. 이걸 알아두는 것이 좋습니다.

문단을 인자로 취하기 만약 어떤 명령이 그 명령이 주어진 위치에서 문단끝까지 인자로 취할 수 있다면 지금 우리가 해본 \mycmd를 중괄호로 문단을 둘러싸지 않고 문단 머리에 두는 것만으로 같은 효과를 낼 수 있겠습니다.

```
\ExplSyntaxOn
\cs_new:Npn \tr_para:w #1 \par
{
    \mycmd { #1 }
    \par
}
\cs_set_eq:NN \trpara \tr_para:w
\ExplSyntaxOff
\trpara
남방의 비는 차갑고 단단하고 찬란한 눈꽃으로 변하는 일이 절대 없다. 식자들은 남방의 비가 단조롭다고
↪ 생각하는데 비 자신도 그것을 불행으로 여기는지 어떤지, 강남의 눈은 그래도 촉촉하고 윤기가 돌아
↪ 아름답기 그지없다. 그것은 아직 눈뜨지 않은 청춘의 소식이며, 건강한 처녀의 살결이다.
```

남방의 비는 차갑고 단단하고 찬란한 눈꽃으로 변하는 일이 절대 없다. 식자들은 남방의 비가 단조롭다고 생각하는데 비 자신도 그것을 불행으로 여기는지 어떤지, 강남의 눈은 그래도 촉촉하고 윤기가 돌아 아름답기 그지없다. 그것은 아직 눈뜨지 않은 청춘의 소식이며, 건강한 처녀의 살결이다.
눈 덮인 들에는 핏빛의 붉은 동백이며, 흰빛에 푸르름이 감도는 외겹 매화며, 경쇠 모양의 진노랑 새양꽃이 있고, 눈 밑에는 아직도 파랗게 언 잡초들이 있다. 나비는 분명 없었다. 꿀벌이 동백꽃과 매화꽃의 꿀을 따러 왔었는지 아닌지, 나는 확실히 기억할 수 없다. 단지 나의 눈앞에는 겨울꽃이 핀 눈 덮인 들판에 바쁘게 날아다니는 수많은 벌들이 보이는 듯, 시끄럽게 붕붕거리는 소리가 들리는 듯했다.

이 명령이 실패할 가능성은 \par 즉 문단 끝을 발견하지 못했을 때입니다. 만약 다른 명령의 인자로부터 이 명령을 건네받는 상황이라면 반드시 \par가 들어가도록 미리 코딩하는 데 주의하여야 합니다. 즉 문단 머리에 mark하는 것으로 원하는 작용을 문단에 대해서 시키는 이런 류의 명령은 "오류가 발생할 가능성"이 일반적인 명령보다 높아진다는 사실을 알고 있어야 합니다.

환경으로 만들기 이번에는 이후에 오는 모든 문단에 이런 걸 붙이라면 어찌해야 할지 생각해봅니다. \everypar라는 프리미티브는 문단을 식자하기 전에 그 문단의 제일 처음에 원하는 토큰을 둘 수 있습니다. 이 선언은 그 이후의 모든 문단에 영향을 끼치지만 다행히 효과는 지역적(local)이므로 범위를 정해주는 것으로 \trpara가 처리하지 못하는 문단이 있어서 발생하는 위험을 줄일 수 있습니다.
이것을 이용하면, LaTeX의 "환경(environment)"이라 불리는 장치를 이용할 수 있음을 알 수 있습니다.

```
\ExplSyntaxOn
\NewDocumentEnvironment { mycmdpara } { +b }
{
    \everypar { \trpara }
    #1
    \par
}{}
\ExplSyntaxOff
```

위와 같이 정의하고 다음 코드를 입력 스트림에 두면

```
\begin{mycmdpara}
```

```
남방의 비는 차갑고 단단하고 찬란한 눈꽃으로 변하는 일이 절대 없다. 식자들은 남방의 비가 단조롭다고
↪ 생각하는데 비 자신도 그것을 불행으로 여기는지 어떤지, 강남의 눈은 그래도 촉촉하고 윤기가 돌아
↪ 아름답기 그지없다. 그것은 아직 눈뜨지 않은 청춘의 소식이며, 건강한 처녀의 살결이다.

눈 덮인 들에는 핏빛의 붉은 동백이며, 흰빛에 푸르름이 감도는 외겹 매화며, 경쇠 모양의 진노랑 새양꽃
↪ 이 있고, 눈 밑에는 아직도 파랗게 언 잡초들이 있다. 나비는 분명 없었다. 꿀벌이 동백꽃과 매화
↪ 꽃의 꿀을 따러 왔었는지 아닌지, 나는 확실히 기억할 수 없다. 단지 나의 눈앞에는 겨울꽃이 핀 눈
↪ 덮인 들판에 바쁘게 날아다니는 수많은 벌들이 보이는 듯, 시끄럽게 붕붕거리는 소리가 들리는 듯했
↪ 다.
\end{mycmdpara}
```

남방의 비는 차갑고 단단하고 찬란한 눈꽃으로 변하는 일이 절대 없다. 식자들은 남방의 비가 단조롭다고 생각하는데 비 자신도 그것을 불행으로 여기는지 어떤지, 강남의 눈은 그래도 촉촉하고 윤기가 돌아 아름답기 그지없다. 그것은 아직 눈뜨지 않은 청춘의 소식이며, 건강한 처녀의 살결이다.
눈 덮인 들에는 핏빛의 붉은 동백이며, 흰빛에 푸르름이 감도는 외겹 매화며, 경쇠 모양의 진노랑 새양꽃이 있고, 눈 밑에는 아직도 파랗게 언 잡초들이 있다. 나비는 분명 없었다. 꿀벌이 동백꽃과 매화꽃의 꿀을 따러 왔었는지 아닌지, 나는 확실히 기억할 수 없다. 단지 나의 눈앞에는 겨울꽃이 핀 눈 덮인 들판에 바쁘게 날아다니는 수많은 벌들이 보이는 듯, 시끄럽게 붕붕거리는 소리가 들리는 듯했다.

원하는 결과를 얻습니다. 약간의 해설을 붙여둡니다.

(a) `+b` 인자 지시자를 써서 문단 내용을 얻어오는 기법을 썼습니다. 이렇게 하지 않고 begin과 end 블럭을 두는 방법으로 정의하는 것은 `\par`를 명시적으로 얻어내기 어려운 경우가 생겨나기 때문에 이 케이스에 한정하여 `+b`가 좋은 방법이 됩니다. 다만 `+b`는 그 내용 중에 verbatim을 포함하면 에러가 발생한다는 사실을 알고 있어야 합니다. 또한 우리가 정의한 `\mycmd`는 수식이나 확장 가능한 명령을 제대로 처리하지 못할 것입니다. 오직 평이한 텍스트만 처리하도록 되어 있으므로 다른 건 욕심내지 마세요. 이런 복잡한 텍스트를 처리하게 하는 게 불가능한가 하면 그것은 아니고요, 단지 여러 경우를 고려해야 하니 복잡해질 따름이죠.

(b) `\end{mycmdpara}` 직전에 `\par`가 반드시 있어야 오류가 발생하지 않을 것은 이미 지적하였습니다. 항상 그러한 것이 아니고 우리가 정의한 `\trpara` 명령이 요구하는 것이기 때문입니다. 그래서 환경의 내용 전체(`#1`) 마지막에 `\par` 하나를 강제로 붙여두고 있습니다.

(c) LaTeX의 environment는 그 자체로 하나의 scope를 이룹니다. 그러므로 굳이 마지막에 `\everypar{}`를 두지 않아도 `\everypar`의 효력이 문단이 종료되면 없어집니다. 그러나 전역적 효과를 가지는 명령을 이 자리에 두는 경우라면 이를 되돌리는 명령을 환경 정의의 <end> 파트에 명기해두어야 할 것입니다.

xparse의 \NewDocumentEnvironment에 대해서 따로 설명하지 않습니다. 패키지 문서를 보세요. 지금 우리가 해본 것은 environment 정의 중에서도 매우 독특한 경우이므로 일반적인 환경 정의와는 살짝 다르다는 사실을 지적해두기로 하겠습니다.

연습문제

기본 4. 다음 표를 완성하고 1em의 크기가 fontsize에 따라 어떻게 달라지는지 조사하여라. 만약 문서 폰트 크기 옵션이 10pt가 아니라 11pt 또는 12pt가 되면 이 값이 어떻게 변하는지도 조사하여라.

우리는 memoir 문서를 작성하고 있으니까(oblivoir) memoir의 폰트 사이즈 명령을 조사해보겠습니다. LaTeX 표준 클래스라면 `\miniscule`과 `\HUGE`가 없고 `\Huge`가 `\HUGE`에 해당한다는 것은 매뉴얼을 읽었으니 다 알고 있겠죠.

```
\ExplSyntaxOn
\clist_set:Nn \l_tmpa_clist { miniscule, tiny, scriptsize, footnotesize,
 ↪  small, normalsize, large, Large, LARGE, huge, Huge, HUGE }

\tl_clear:N \l_tablines_tl
\clist_map_inline:Nn \l_tmpa_clist
{
    \tl_put_right:Nn \l_tablines_tl
    {
        \texttt { \bs #1 } &
        \tl_use:c { #1 }
        \dim_set:Nn \l_tmpa_dim { 1em }
        { \normalsize \dim_use:N \l_tmpa_dim }
        \newlinehline
    }
}

\begin{tabular}[b]{r|r }
\hline
font~size~command & value~of~em \\ \hline
\l_tablines_tl
\end{tabular}
\ExplSyntaxOff
```

font size command	value of em
\miniscule	5.0pt
\tiny	6.0pt
\scriptsize	7.0pt
\footnotesize	8.0pt
\small	9.0pt
\normalsize	10.0pt
\large	10.95pt
\Large	12.0pt
\LARGE	14.4pt
\huge	17.28pt
\Huge	20.74pt
\HUGE	24.88pt

line 2:
왜 `\miniscule, \tiny`와 같이 clist에 넣지 않고 `miniscule, tiny`와 같이 넣어두는가 하면 백슬래시를 붙여두면 그것은 "확장 가능한 매크로"가 되어 버려서 언제 확장되어 버릴지 모르기 때문입니다. 이름만 넣어두었다가 필요할 때 `\use:c` 또는 `\tl_use:c`로 효력이 생기게 하는 것이 좋습니다. `\use:c`의 용법에 대해서는 번외편 Special Course I을 참고하세요.
line 12:
10행에서 폰트 크기 명령이 효력을 발생하였습니다. 따라서 `\dim_use:`를 그냥 쓰면 폰트 크기 명령의 효력이 미쳐서 크게 또는 작게 찍힐 것입니다. 크기 자체만 표시하기를 원하고 그것이 폰트 사이즈대로 찍히는

것을 원치 않았기 때문에 {\normalsize...라고 하여 찍은 것입니다. 중괄호는 범위를 지정한 것으로 이해하세요.

문서 폰트 사이즈 옵션 현재 문서의 폰트 사이즈 명령에 대해서는 이것으로 되었습니다. 그런데

```
\documentclass[11pt]{oblivoir}
```

이런 경우에 어찌 되는지 물었으니 이 부분을 해결해야 합니다.
제일 쉬운 것은 .tex 파일을 하나씩 마련하여 거기서 나오는 결과를 조사하는 것입니다. 11포인트를 위하여 다음과 같은 fsize-11.tex을 만들어보겠습니다.

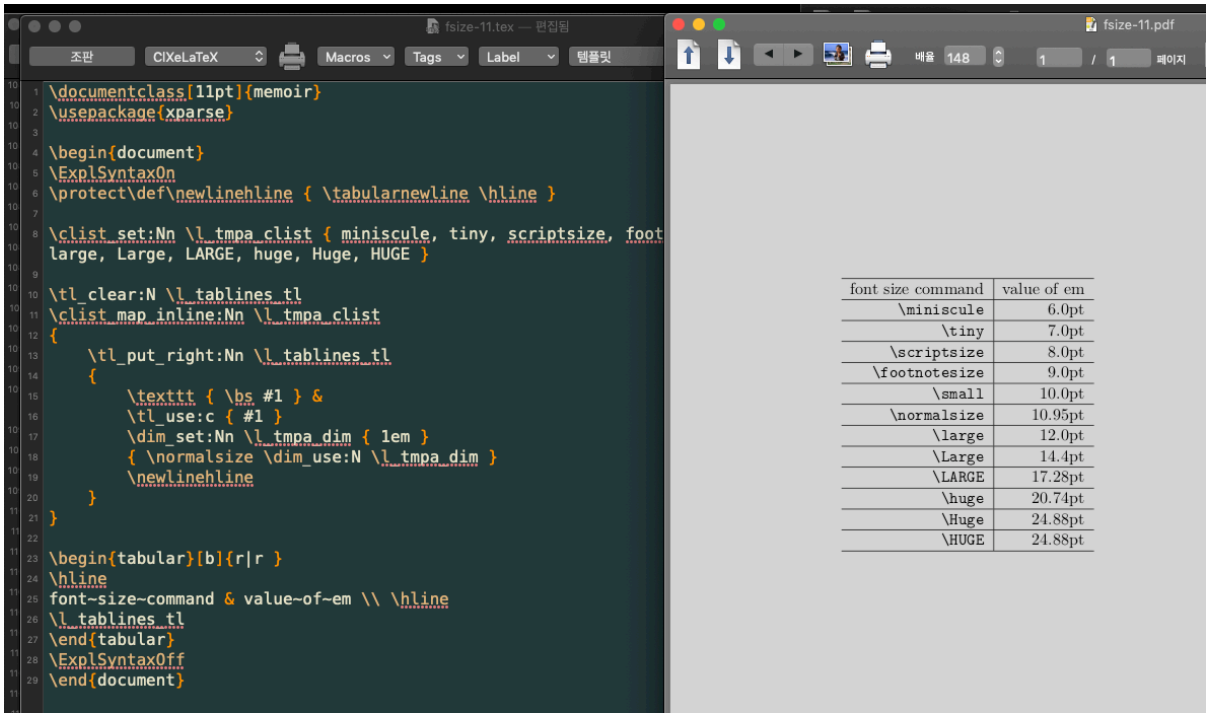
```
\documentclass[11pt]{memoir}
\usepackage{xparse}

\begin{document}
\ExplSyntaxOn
\protect\def\newlinehline { \tabularnewline \hline }

\clist_set:Nn \l_tmpa_clist { miniscule, tiny, scriptsize, footnotesize,
 ↪  small, normalsize, large, Large, LARGE, huge, Huge, HUGE }

\tl_clear:N \l_tablines_tl
\clist_map_inline:Nn \l_tmpa_clist
{
    \tl_put_right:Nn \l_tablines_tl
    {
        \texttt { \bs #1 } &
        \tl_use:c { #1 }
        \dim_set:Nn \l_tmpa_dim { 1em }
        { \normalsize \dim_use:N \l_tmpa_dim }
        \newlinehline
    }
}

\begin{tabular}[b]{r|r }
\hline
font~size~command & value~of~em \\ \hline
\l_tablines_tl
\end{tabular}
\ExplSyntaxOff
\end{document}
```

이것을 저장하고 다음과 같이 컴파일해봅니다. (이 문서에는 한글이 한 글자도 들어가지 않기 때문에 그냥 memoir 클래스로 했습니다.)

```
$ pdflatex fsize-11|
```

그러면 그림과 같은 결과를 얻을 수 있습니다. 이것을 12pt에 대해서도 시행하면 되는 것입니다.

표 만들기 그런데 숙제를 낼려니 이 수치들을 모두 베껴와야 하잖아요? 보고 적는 거야 어려울 거 없지만 뭔가 없어보여서... 고급진(?) 방법이 없나 생각해봅니다.
이 수치들을 내가 일일이 보고 베끼지 않아도, LaTeX이 컴파일하면서 그 결과를 clist에 넣어 전달해주면 좋을텐데요. 그렇게 하게 하려면 (아직 배우지 않은) file 데이터타입을 이용할 수 있습니다. 어차피 file은 다음다음 주의 학습과제니까 여기서 간단히 예를 들어두고 지나가겠습니다.
일단 fsize-11.tex을 다음처럼 작성합니다.




```
\documentclass[11pt]{memoir}
\usepackage{xparse}

\begin{document}
\ExplSyntaxOn

\tl_set:Nn \l_size_tl { eleven }

\clist_set:Nn \l_tmpa_clist { miniscule, tiny, scriptsize,
    footnotesize, small, normalsize, large, Large, LARGE,
    huge, Huge, HUGE}

\clist_clear:N \l_tmpb_clist

\clist_map_inline:Nn \l_tmpa_clist
{
    \tl_use:c { #1 }
    \tl_set:Nx \l_tmpa_tl { \dim_eval:n { 1em } }
    \clist_put_right:NV \l_tmpb_clist \l_tmpa_tl
}

\clist_use:Nn \l_tmpb_clist { ,~ }

\iow_new:N \l_out_iow
\iow_open:Nn \l_out_iow { \jobname.clist.txt }
\iow_now:Nn \l_out_iow { \clist_set:cn }
\iow_now:Nx \l_out_iow { { l _ \l_size_tl _ clist } }
\iow_now:Nx \l_out_iow { { \clist_use:Nn \l_tmpb_clist {,~ } } }
\iow_close:N \l_out_iow

\ExplSyntaxOff
\end{document}
```

이걸 컴파일하면 작업 폴더에 `fsize-11.clist.txt`라는 파일이 생겨 있습니다. 에디터로 열어보면 다음과 같습니다.

```
\clist_set:cn
{l_eleven_clist}
{6.0pt, 7.0pt, 8.0pt, 9.0pt, 10.0pt, 10.95pt, 12.0pt, 14.4pt, 17.28pt,
 ↪  20.74pt, 24.88pt, 24.88pt}
```

12pt문서를 위해서 fsize-12.tex을 만들 때는 다음 부분이 고쳐져야 하겠지요.

```
\documentclass[12pt]{memoir}
```

여기와

```
\tl_set:Nn \l_size_tl { twelve }
```

여기 두 군데입니다.
마찬가지로 컴파일하면 fsize-12.clist.txt가 생겨나야 합니다. 생성되는 파일 이름을 바꾸고 싶다면 \jobname.clist.txt를 원하는 이름으로 고치세요. 그러나 tex 파일 자체와 확장자까지 같은 이름이 되게 하면 안 됩니다. 확장자는 txt가 좋다고 생각합니다.
작업 폴더에 fsize-11.clist.txt와 fsize-12.clist.txt가 있으니까 이제 표를 만들어보겠습니다. 이참에 fsize-10.clist.txt도 같이 만들어두지요.
먼저, 파일을 \input하면 해당 clist가 잘 인식되는지 보기로 합니다.

```
\ExplSyntaxOn
\input{fsize-10.clist.txt}
\input{fsize-11.clist.txt}
\input{fsize-12.clist.txt}

\clist_use:Nn \l_ten_clist {,~} \par
\clist_use:Nn \l_eleven_clist {,~} \par
\clist_use:Nn \l_twelve_clist {,~}
\ExplSyntaxOff
```

5.0pt, 6.0pt, 7.0pt, 8.0pt, 9.0pt, 10.0pt, 10.95pt, 12.0pt, 14.4pt, 17.28pt, 20.74pt, 24.88pt
6.0pt, 7.0pt, 8.0pt, 9.0pt, 10.0pt, 10.95pt, 12.0pt, 14.4pt, 17.28pt, 20.74pt, 24.88pt, 24.88pt
7.0pt, 8.0pt, 9.0pt, 10.0pt, 10.95pt, 12.0pt, 14.4pt, 17.28pt, 20.74pt, 24.88pt, 24.88pt, 24.88pt

잘 들어왔군요. 이제 표 그리는 일만 남았으니 어렵지 않겠죠.

```
\ExplSyntaxOn
\protected\def\newlinehline{\tabularnewline \hline}
\input{fsize-10.clist.txt}
\input{fsize-11.clist.txt}
\input{fsize-12.clist.txt}

\seq_set_from_clist:Nn \l_tmpa_seq { miniscule, tiny, scriptsize,
    footnotesize, small, normalsize, large, Large, LARGE,
    huge, Huge, HUGE }

%%% headline
\tl_set:Nn \l_tablines_tl { & 10pt & 11pt & 12pt \newlinehline }

\seq_indexed_map_inline:Nn \l_tmpa_seq
{
    \tl_put_right:Nn \l_tablines_tl { #2 & }
    \tl_put_right:Nx \l_tablines_tl { \clist_item:Nn \l_ten_clist { #1 } }
    \tl_put_right:Nn \l_tablines_tl { & }
    \tl_put_right:Nx \l_tablines_tl { \clist_item:Nn \l_eleven_clist { #1 } }
    \tl_put_right:Nn \l_tablines_tl { & }
```

```
    \tl_put_right:Nx \l_tablines_tl { \clist_item:Nn \l_twelve_clist { #1 } }
    \tl_put_right:Nn \l_tablines_tl { \newlinehline }
}

\begin{tabular}{c|r|r|r}
\hline
\l_tablines_tl
\end{tabular}
\ExplSyntaxOff
```

	10pt	11pt	12pt
miniscule	5.0pt	6.0pt	7.0pt
tiny	6.0pt	7.0pt	8.0pt
scriptsize	7.0pt	8.0pt	9.0pt
footnotesize	8.0pt	9.0pt	10.0pt
small	9.0pt	10.0pt	10.95pt
normalsize	10.0pt	10.95pt	12.0pt
large	10.95pt	12.0pt	14.4pt
Large	12.0pt	14.4pt	17.28pt
LARGE	14.4pt	17.28pt	20.74pt
huge	17.28pt	20.74pt	24.88pt
Huge	20.74pt	24.88pt	24.88pt
HUGE	24.88pt	24.88pt	24.88pt

나중에 file에 대하여 배운 후에는 셸을 열고 `fsize-11.tex`을 컴파일하고 하는 과정까지 모두 일괄처리하는 방법을 알게 됩니다.

연습문제

기본 1. 심심해진 철수는 idle 프로그램으로 다음과 같은 것을 하면서 놀았다.

```
>>> import random
>>> total = 10000
>>> ev = 0
>>> for i in range(total):
        x = random.random()
        y = random.random()
        if x*x+y*y <= 1:
                ev += 1


>>> pi = ev/total * 4
>>> print (pi)
3.124
>>>
```

이것을 본 영희가 "나는 expl3로 할 수 있어"라고 하였다. 과연 할 수 있었을까?

그리 어려운 문제가 아니므로 다음 코드만 보이겠습니다. 예시 python의 10000번은 expl3로 안 되지는 않는데 기다리는 시간이 조금 지루해서 그냥 100번으로 했습니다.

```
\ExplSyntaxOn

\fp_new:N \l_x_fp
\fp_new:N \l_y_fp
\int_new:N \l_ev_int

\int_step_inline:nn { 100 }
{
    \fp_set:Nn \l_x_fp { rand () }
    \fp_set:Nn \l_y_fp { rand () }

    \fp_compare:nT { \l_x_fp ** 2 + \l_y_fp ** 2 <= 1 }
    {
        \int_incr:N \l_ev_int
    }
}

\[ \pi \simeq \fp_eval:n { \l_ev_int / 100 * 4 } \]

\ExplSyntaxOff
```

$$\pi \simeq 3.16$$

연습문제

기본 2. 등거리 random walk을 모사하려고 한다. 회전각을 난수적으로 얻어서 20회 정도 진행한 궤적을 Ti*k*Z로 나타내어라. 진행 거리는 0.5cm이다.

지난 주에 풀었던 문제와 거의 똑같은 거죠. 단지 회전각이 random이라는 것만 다르고 진행거리가 일정해서 더 쉬운 문제입니다.

```
\ExplSyntaxOn

\tl_set:Nn \l_forward_tl { 0.5 }

\tl_clear:N \l_tmpa_tl

\tl_set:Nn \l_tmpa_tl { (0,0) }

\int_step_inline:nn { 20 }
{
    \tl_put_right:Nx \l_tmpa_tl
    {
        -- ( [turn] \fp_eval:n { 360 * rand () }\c_colon_str \l_forward_tl )
    }
}

\begin{tikzpicture}
\node at (0,0) [red] {\textbullet};
\draw \l_tmpa_tl ;
\end{tikzpicture}

\ExplSyntaxOff
```

[No. 7] **Cat in a Box** ____년 __월 __일

연습문제

기본 1. 다음 그림은 철수가 작성하고 있는 문서의 소스이다. 표시된 부분이 TEX으로 처리될 때 해당 위치가 hmode 상태인지 vmode 상태인지를 적시하여라.

```
\begin{document}
(1)
혁명의 길은 파괴부터 개척할지니라.
그러나 파괴만 하려고 파괴하는 것이 아니라 건설하려고 파괴하는 것이니,
(2)     (3)
\hbox{만일 건설할 줄을 모르면} 파괴할 줄도 모를 지며,
파괴할 줄을 모르면 건설할 줄도 모를지니라. \\ (4)

건설과 파괴가 다만 형식상에서 보아 구별될 뿐이요, 정신상에서는 파괴가
곧 건설이니 이를테면 우리가 일본 세력을 파괴하려는 것이 제1은, 이족통
치를 파괴하자 함이다.
(5)
(6)

왜? 〈조선〉이란 그 위에 〈일본〉이란 이민족 그것이 전제(專制)하여 있으
니,
이족 전제의 밑에 있는 조선은 고유적 조선이 아니니, 고유적 조선을 발견
하기 위하여 이족통치를 파괴함이니라.
```

(1): vmode, (2): vmode, (3): hmode, (4): hmode, (5): hmode, (6): vmode

```
\ExplSyntaxOn
\NewDocumentCommand \checkmode { }
{
    \mode_if_horizontal:TF { [H] }
    {
        \mode_if_vertical:T { [V] }
    }
}
\ExplSyntaxOff

\checkmode 혁명의 길은 파괴부터 개척할지니라.

\checkmode\hbox{\checkmode 만일 건설할 줄을 모르면} 파괴할 줄도
모를지니라. \\ \checkmode

이족통치를 파괴하자 함이다.
\checkmode

\checkmode

왜?
```

[V]혁명의 길은 파괴부터 개척할지니라.
[V][H]만일 건설할 줄을 모르면 파괴할 줄도 모를지니라.
[H]
이족통치를 파괴하자 함이다. [H]

[V]
왜?

연습문제

기본 2. 다음 페이지의 그림은 어떤 시험의 문제지 한 면이다. 문항 번호는 문제에 대하여 큰 글자로 왼쪽으로 튀어나오게 조판되어 있다. 이 페이지를 조판해보아라. 필요하다면 문제번호, 문제, 선택지 문항을 적절한 box에 넣어서 처리하여라. 모양이 반드시 예시된 그림과 동일하지 않아도 상관없다. 또한 그래프는 화면을 캡처하여 그래픽 파일로 활용하여도 좋다.

정답과 해설을 생략합니다. (숙제를 제출하면 그것을 소개하고 코멘트할 계획이었으나...)

연습문제

기본 3. 같은 논문의 예제 13, 예제 16, 예제 18을 expl3로 해보아라.

예제 13 `\usebox`와 `\copy`, `\unhcopy` 비교.
예제에서와 같이 마지막의 `\unhcopy`가 박스 자체가 아니라 박스의 내용물을 unpack하는 것임을 보이기 위하여 `\hsize=3.5cm`로 하였습니다.

```
\hsize=3.5cm
\newsavebox\mybox
\sbox\mybox{A B C }
\newcommand\UB{\usebox\mybox}
\newcommand\BC{\copy\mybox}
\newcommand\UC{\unhcopy\mybox}
\UB\UB\UB\UB\par
\BC\BC\BC\BC\par
\UC\UC\UC\UC\par
```

A B C A B C A B C A B C
A B C
A B C
A B C
A B C
A B C A B C A B C A B
C

위의 예에서 `\usebox` 매크로는 만약 현재 mode가 vertical이면 `\leavevmode` 하고 `\copy`하는 명령입니다.

```
\def\usebox#1{\leavevmode\copy#1\relax}
```

expl3의 `\box_use:N` 함수는 `\copy`와 동일한 명령이므로 `\usebox`에 해당하는 명령을 다음과 같이 작성하였습니다.
참고로, TEX에서 `\box`<*box register*> 명령과 `\copy`<*box register*> 명령의 차이는 `\box_use_drop:N`과 `\box_use:N`의 차이와 같습니다.

```
\ExplSyntaxOn
\hsize=3.5cm
\cs_new:Npn \usebox_cmd:N #1
{
    \mode_if_vertical:T { \mode_leave_vertical: } \box_use:N #1
}
\box_new:N \l_mybox_box
\hbox_set:Nn \l_mybox_box { A~B~C~ }
\cs_new:Nn \UB: { \usebox_cmd:N \l_mybox_box }
\cs_new:Nn \BC: { \box_use:N \l_mybox_box }
\cs_new:Nn \UC: { \hbox_unpack:N \l_mybox_box }
\int_step_inline:nn { 4 } { \UB: } \par
\int_step_inline:nn { 4 } { \BC: } \par
\int_step_inline:nn { 4 } { \UC: } \par
\ExplSyntaxOff
```

A B C A B C A B C A B C
A B C
A B C

```
A B C
A B C
A B C A B C A B C A B
C
```

예제 16 `\vbox`와 `\vtop`.

```
\hsize=2cm
A\vbox{B\par C}D\vtop{E\par F}G\par
A\vbox{\hbox{B}\par\hbox{C}}D%
 \vtop{\hbox{E}\par\hbox{F}}G\par
A\vbox{\mbox{B}\par\mbox{C}}D%
 \vtop{\mbox{E}\par\mbox{F}}G
```

```
  B
AC          DE          G
              F
  B
ACDEG
    F
  B
AC          DE          G
              F
```

expl3에도 "즉시 box" 함수들이 있습니다. 각각, `\hbox:n`, `\vbox:n`, 그리고 `\vbox_top:n`입니다.

```
\hsize=2cm
\ExplSyntaxOn
A \vbox:n { B \par C } D \vbox_top:n { E \par F } G \par

A \vbox:n { \hbox:n { B } \par \hbox:n { C } } D
  \vbox_top:n { \hbox:n { E } \par \hbox:n { F } } G \par

A \vbox:n { \mbox { B } \par \mbox { C } } D
  \vbox_top:n { \mbox { E } \par \mbox { F } } G
\ExplSyntaxOff
```

```
  B
AC          DE          G
              F
  B
ACDEG
    F
  B
AC          DE          G
              F
```

이 예제는 `\vbox`, `\hbox`, `\mbox`의 차이를 이해하라는 것입니다. `\mbox`는 `\leavevmode`가 있는 `\hbox`입니다.

예제 18 `\settowidth` 테크닉

```
\newlength\mylen
\settowidth\mylen{C}%
A\parbox[b]{\mylen}{B\par C}D%
```

```
 \settowidth\mylen{E}%
 \parbox[t]{\mylen}{E\par F}G
```

```
 B
ACDEG
    F
```

```
\ExplSyntaxOn
\cs_new:Npn \set_to_width:Nn #1 #2
{
    \hbox_set:Nn \l_tmpa_box { #2 }
    \dim_set:Nn #1 { \box_wd:N \l_tmpa_box }
}

\set_to_width:Nn \l_tmpa_dim { C }
A \parbox[b] { \l_tmpa_dim } { B \par C } D

\set_to_width:Nn \l_tmpa_dim { E }
  \parbox[t] { \l_tmpa_dim } { E \par F } G
\ExplSyntaxOff
```

```
 B
ACDEG
    F
```

\settowidth는 위와 같이 새로 정의하지 않고 그대로 써도 좋습니다.

연습문제

기본 4. 문단을 파고드는 그림을 조판하려 한다. 그림 이전에 문단의 2행이 오고 그림은 문단의 오른쪽에 놓인다. 이런 형식의 문단을 그림과 함께 조판하여 보아라. 그림은 mwe 패키지가 제공하는 example-image.png를 이용하되,(즉, 그냥

```
\includegraphics{example-image}
```

로 조판이 가능하다. 이 그림이 실제 어느 위치에 있는지는 신경쓸 필요 없다.) 그림 크기는 width=2.8cm로 하고 텍스트 본문의 행간격은 1로 하여라. wrapfig 류의 외부 패키지 사용은 금지한다.

그림의 높이는 문단의 몇 줄이나 차지할까? 문제에서 `width=2.8cm`로 하라고 했습니다. 그리고 행간격을 1로 하라고 하는데 행간격이 1일 때 `\normalsize` 폰트이고 10pt 문서라면 `\baselineskip`은,

```
\begin{SingleSpace}
\ExplSyntaxOn
\dim_eval:n { \baselineskip }
\ExplSyntaxOff
\end{SingleSpace}
```

12.0pt

입니다.
그림을 적당한 박스에 넣고 높이를 잰 다음 (그림 박스는 depth가 0이므로 depth에 대해 걱정할 필요는 아직 없습니다) 이 길이를 저 수치로 나누어보겠습니다.

```
\begin{SingleSpace}
\ExplSyntaxOn
\hbox_set:Nn \l_tmpa_box { \includegraphics [width=2.8cm ] { example-image }
 ↪  }
\dim_set:Nn \l_tmpa_dim { \box_ht:N \l_tmpa_box }
\dim_use:N \l_tmpa_dim \par %% 출력

\int_set:Nn \l_tmpa_int { \fp_eval:n { round ( \dim_ratio:nn { \l_tmpa_dim }
 ↪  { \baselineskip } ) } }
\int_use:N \l_tmpa_int %% 출력

\ExplSyntaxOff
\end{SingleSpace}
```

59.75066pt
5

이 계산의 결과는 5줄이라고 나옵니다. 그런데 우리는 그림의 아래와 위에 조금 여유분을 둘 생각이니까 간단히 여기에 1을 더한 숫자로 그림이 차지할 수직 공간의 행수를 설정하겠습니다.
이 숫자를 `\l_vlinesimage_int`에 넣고 `\parshape`를 설정합니다. parshape의 행 수는 처음에 오는 2행 + 그림이 차지하는 공간 6행 + 원래대로 되돌리는 1행 해서 모두 9행입니다. 이 숫자들은 모두 조작 가능하지만 여기서는 그냥 써넣겠습니다.
parshape를 주는 길이는 그림이 차지할 3cm (2.8cm + 2mm)를 비우는 것인데 여기서 2mm는 그림과 텍스트 사이의 간격이 될 것입니다. 이 길이 역시 계산가능하지만 여기서는 그냥 써 넣겠습니다.

```
\begin{SingleSpace}
\ExplSyntaxOn
\int_new:N \l_vlinesimage_int
\int_set:Nn \l_vlinesimage_int { 6 }

\tl_set:Nx \l_tmpa_tl {
    0pt~\textwidth ~
    0pt~\textwidth ~
}

\int_step_inline:nn { \l_vlinesimage_int }
{
    \tl_put_right:Nx \l_tmpa_tl {
        0pt~\dim_eval:n { \textwidth - 3cm } ~
    }
}

\tl_put_right:Nx \l_tmpa_tl { 0pt~\textwidth }

\tl_put_left:Nn \l_tmpa_tl { 9~ }

\exp_last_unbraced:Nx \parshape \l_tmpa_tl
\ExplSyntaxOff
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
 ↪ tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
 ↪ veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
 ↪ commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
 ↪ velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
 ↪ occaecat cupidatat non proident, sunt in culpa qui officia deserunt
 ↪ mollit anim id est laborum. Sed ut perspiciatis unde omnis iste natus
 ↪ error sit voluptatem accusantium doloremque laudantium, totam rem
 ↪ aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto
 ↪ beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia
 ↪ voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni
 ↪ dolores eos qui ratione voluptatem sequi nesciunt.
\end{SingleSpace}
```

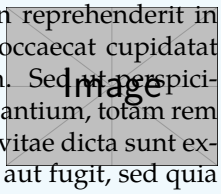
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

그림이 들어가는 박스 그림 박스를 저 위치에 가져다 놓는 것을 해봅니다.

```
\begin{SingleSpace}
\ExplSyntaxOn
\hbox_set:Nn \l_tmpa_box
{
    \hspace { \dim_eval:n { \textwidth - 2.8cm } }
    \includegraphics [width=2.8cm] { example-image }
}
\vbox_set:Nn \l_tmpb_box {
```

```
    \vskip 1.5\baselineskip
    \box_use:N \l_tmpa_box
}
\box_set_wd:Nn \l_tmpb_box { 0pt }
\box_set_ht:Nn \l_tmpb_box { 0pt }

\mode_leave_vertical: \box_use:N \l_tmpb_box
\ExplSyntaxOff
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
 ↪ tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
 ↪ veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
 ↪ commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
 ↪ velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
 ↪ occaecat cupidatat non proident, sunt in culpa qui officia deserunt
 ↪ mollit anim id est laborum. Sed ut perspiciatis unde omnis iste natus
 ↪ error sit voluptatem accusantium doloremque laudantium, totam rem
 ↪ aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto
 ↪ beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia
 ↪ voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni
 ↪ dolores eos qui ratione voluptatem sequi nesciunt.
\end{SingleSpace}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.



처음에 그림을 hbox에 넣을 때 그림 왼쪽에 충분한 공간 (textwidth - 그림 width)을 주어서 그림이 문단의 오른쪽 끝에 가도록 했습니다.
그 다음에 이것을 vbox에 넣으면서 1.5baselineskip 만큼을 그림 윗쪽에 두어서 2행을 건너뛰도록 했는데 실제로 여기서 주고 싶은 값은 2.5였을 텐데 왜 1.5로 했느냐 하면 시작점이 첫 줄의 baseline이기 때문입니다. 그래서 1을 빼준 거죠.
최종적으로 만들어진 박스가 문단 모양을 흐트리지 않도록 width와 height를 모두 0pt로 만들었습니다.

해결 이 둘을 모두 합친 결과를 다음 페이지에 보였습니다.
이 해결책에는 몇 가지 문제가 있는데요, 첫째는 문단 중간에 페이지가 나누어질 때 그림은 함께 나누어지지 않는다는 것인데 이것은 문제라기보다 원래 그런 것이라고 보아야 할 것입니다. 둘째는 문단이 짧아서 그림을 둘러쌀 충분한 텍스트를 제공하지 못할 때 이어지는 문단에도 그림을 위한 공간을 마련해야 할텐데 그런 문제가 고려되고 있지 않다는 것입니다. 그러나 위에 제시한 아주 간단한 코딩으로 이런 모양을 쉽게 만들 수 있다는 것을 경험해보는 것이 목적이므로 충분히 달성되었다고 하겠습니다.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.