예제로 배우는 expl3

이엑스피엘쓰리 스타디 그룹

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

이 문서에 관하여

한국텍학회와 한글텍사용자그룹의 활동으로서 2019년 7월~8월간 진행된 "Expl3 스터디 그룹"에서 공부한 자료 모음이다. 자료는 "예제", "강의", "연습문제", "연습문제 정답과 해설", "숙제에 관한 코멘트"로 이루어져 있었으며 공개 자료는 그 가운데 예제와 강의 및 연습문제 부분을 포함한다.
2019년 11월 이후 Expl3이 LaTeX 포맷에 포함되는 과정에서 변경된 부분이 있어, 2020년 2월에 해당 부분을 고친 개정판을 작성하였다. 2020년 9월에 수정된 expl3를 반영하여 더 고쳤다. 2022년 5월에 약간의 수정을 더하였다.

예제

인자를 하나 받아서 "Hello, #1!'' 형식으로 출력하는 명령 `\hellocmd`를 작성하여라.

입력: `\hellocmd{world}`
출력: Hello, world!

xparse xparse에서 다음 명령은 이미 알고 있다고 가정한다. 만약 다음 명령이 무엇을 하려는 것인지 잘 모르겠다면 xparse 문서를 반드시 숙지하여야 한다.

- `\NewDocumentCommand`.
- `\RenewDocumentCommand`.
- `\IfNoValueTF`.
- `\IfBooleanTF`.
- 인자형 지시자: m, o, O, s.

interface3.pdf 이 문서는 expl3의 (거의 유일하고 완전한) reference이다. 항상 열어두고 작업하는 것이 옳다. `texdoc interface3`.

이제 주어진 문제를 풀어보자.

```
\ExplSyntaxOn
\NewDocumentCommand \hellocmd { m }
{
    Hello, ~ #1 !
}
\ExplSyntaxOff
```

`\ExplSyntaxOn`과 `\ExplSyntaxOff`로 expl3 언어로 코딩하는 부분임을 표시한다. 이 범위 안에서는

1. 스페이스는 모두 무시된다.
2. :과 _가 명령의 일부로 쓰인다(즉, letter이다).

그러므로 행끝 문자를 없애기 위해서 %를 행 끝에 분여야 했던 불편이 없다. 그 대신 스페이스를 입력 문자열에 남겨야 할 적에 반드시 ~(틸데)로 그것을 명시적으로 표시해주어야 한다.
위의 명령을 실행하면 다음처럼 된다.

```
\hellocmd{world}
```

Hello, world!

이번에는 다음과 같은 코드를 생각해보자.

```
\ExplSyntaxOn
\cs_new:Npn \hello_fn:n #1
```

```
{
    Hello,~ #1 !
}

\NewDocumentCommand \hellocmd { m }
{
    \hello_fn:n { #1 }
}
\ExplSyntaxOff
```

이 코드는 문서 명령이 어떤 것인지를 명백하게 보여준다. 문서 명령 `\hellocmd`가 하는 일은 인자를 하나 받아서 expl3로 작성된 "함수'' `\hello_fn:n`에 넘겨주는 일만을 한다. 실제로 어떤 작용을 하는 것은 이 함수가 하게 되는 것이다. 그러나 `\begin{document}` 이후에 `\hello_fn:n`이라는 명령을 쓸 수는 없기 때문에 문서 명령 `\hellocmd`를 정의하게 되었다.
한편, 우리가 넘겨받은 인자는 평범한 알파벳으로 이루어진 문자열(토큰열)이다. 앞으로 공부하게 될 중요한 자료형은 다음과 같은 것이 있다.

token list 가장 기본적인 문자열이다. 매크로나 그룹도 하나의 토큰으로 간주한다. 토큰열에서 모든 문자는 그 카테고리 코드를 보존하고 있다.

string 토큰열과 같지만 카테고리 코드가 모두 12(other)로 간주된다. 따라서 매크로는 그냥 문자열일 뿐이고 실행(확장)되지 않는다.

integer 정수형. 보통의 형검사를 하는 언어에서의 int와 매우 유사하다.

floating point . 부동소숫점 실수형. 유효숫자가 16자리인 실수이다.

dimension TEX의 길이값 형식으로서 단위(sp, pt, mm, etc.)를 포함한다.

skip TEX의 skip 형식으로서 둘 이상의 dim으로 구성된다.

keys <*key*> = <*value*> 형식의 데이터.

여기에 다음과 같은 리스트 형식의 자료형이 존재한다.

sequence expl3 특유의 리스트 자료형이다. 여러 개의 아이템을 일관되게 다루기 위한 것으로 stack처럼 활용할 수 있다.

clist comma separated list. seq의 특별한 경우로서 각 아이템이 ,로 구분된 것이다. 간략화한 seq라고 생각하면 된다.

property list key=value 형식의 리스트이다.

이밖에, file, regex, sys, quark, cs, box, coffin 등의 자료형이 더 있지만 천천히 배워나갈 것이다.

연습삼아서 인자로 받은 문자열을 tl 변수에 넣어서 활용하는 방법을 생각해보자.

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

`\cs_new:Npn`은 새로운 함수(cs)를 정의하라는 명령으로서 자주 쓰일 것이므로 이 문법을 잘 보아두어야 한다. 이 자체가 하나의 함수이다. N은 control sequence, p는 parameter, n은 "중괄호로 둘러싸인 토큰열''을 가리킨다. 이 인자형 지시자는 매우 중요한데, 지금 다 알아둘 필요는 없지만 다음 몇 가지는 숙지해두자.

n 가장 많이 쓰이는 인자형이다. 중괄호로 묶어서 전달되는 토큰열. {...}가 하나의 n에 해당한다.

N 한 개의 매크로를 가리킨다. 예를 들면 `\l_tmpa_tl` 자체가 하나의 N이다.

o, x, f, e 인자의 확장형을 가리키는데 expansion 문제는 나중에 천천히 다루게 된다.

V 한 개의 매크로를 가리키지만 그 매크로 자체를 전달하는 것이 아니라 그 값(value)을 넘겨받는다는 의미이다.

따라서 `\tl_set:Nn`은 그 뒤에 차례로 한 개의 매크로와 한 개의 중괄호 범위가 올 것임을 예상할 수 있다. 이 함수의 의미는 n으로 주어지는 토큰열을 N이라는 매크로에 할당하라는 것이다. 당연히 N은 tl 자료형이어야 한다.
이 때 `\l_tmpa_tl`은 어떤 데이터를 저장할 매크로인데, 이를 "변수"라고 한다. 변수는 어떤 자료형인지를 명시하는 것이 좋다. l은 이것이 지역변수임을 의미하고 tmpa는 고유한 이름이며 tl은 데이터 타입이다.
이름이 `tmpa`인 것과 `tmpb`인 것은 expl3 자체에 정의된 scratch 변수이다. 즉 이 변수명을 선언하지 않아도 바로 사용할 수 있다. 스크래치 변수는 l과 g에 대하여 두 개씩 있어서 모두 네 개가 각 데이터별로 정의되어 있다. 자신만의 변수를 만들려면 `\tl_new:N`으로 변수를 선언해야 할 때가 있다.
위의 코드에서 둘째 줄 `\l_tmpa_tl`은 그 자체로 그 위치에 이 변수의 내용을 식자한다. 이렇게 하는 것을 "입력문자열에 남긴다"고 표현한다. 그리고 그것이 이 함수의 반환값 비슷한 것이 된다는 점에 유의하라. expl3는 명시적인 반환값은 별도로 존재하지 않고(따라서 엄밀한 의미의 function이 아니다), 입력 문자열에 어떤 것을 남김으로써 반환값처럼 처리하게 한다. 요컨대, expl3에는 `return` 문이 없다.

용어

문서 명령 Expl3로 작성하는 control sequence들은 _(underscore)와 :(colon)을 포함하므로 문서 중에 이를 사용할 수 없다. expl3를 이용하여 작성한 함수를 문서 중에 사용하기 위하여 만드는 매크로를 "문서 명령(document commands)"라고 부른다. 문서 명령을 작성하도록 도와주는 것이 xparse이며 expl3와 필수적으로 함께 쓰여야 한다.

입력 문자열 *input stream*. TEX에게 전해주는 토큰(열)을 가리키는 말로 쓴다.

인자형 지시자 expl3 함수의 일부로서 그 함수가 흡수(absorb)할 인자의 형식을 미리 지정한다. 참고로, xparse의 인자형 지시자(m, o, 등)는 expl3의 인자형 지시자와 기능과 목적이 다 다르므로 혼동하면 안 된다.

자료형 expl3의 data type.

함수 인자를 취하여 일정한 작용을 하도록 정의한 expl3의 메크로(데이터) 형식

변수 데이터를 저장하고 있는 매크로

범위 `\begingroup`에서 `\endgroup`까지, 즉 {에서 }까지를 가리킨다. 지역(local) 변수는 이 범위 내에서만 유효하다. 범위와 상관없이 문서 전체에 유효한 변수를 전역적(global)이라고 한다.

연습문제

기본 1. 문제에서 제시한 `\hellocmd`에서 인자로 주어진 이름을 이탤릭체로 식자하도록 해보아라.

발전 2. 문제에서 제시한 `\hellocmd`를, 넘어온 인자의 첫 글자를 대문자로 바꾸어 출력하도록 작성하여라.

입력: `\Hellocmd{world}`
출력: Hello, *world*! 또는 Hello, World!

첫 숙제에 관하여

첫 글자만 대문자로 바꾸는 방법에 대한 질문이 있었습니다.

TEX과 LATEX의 방법 단어가 인자로 들어왔을 때 첫글자와 그 뒷글자로 분리하는 전통적 방법은 다음과 같습니다.

```
\def\mytmpfn#1#2\end{\def\tmpA{#1}\def\tmpB{#2}}
\def\headandtail#1{\expandafter\mytmpfn #1\end}

\headandtail{dream}
[\tmpA] (\tmpB)
```

[d] (ream)

\uppercase를 사용할 때는 그 뒤에 letter가 아니면 효과가 없습니다. 따라서 매크로가 들어올 적에 이를 확장해주어야 하는데 그러면 대략 다음과 같은 모양이 됩니다.

```
\def\mytmpfn#1#2\end{\def\tmpA{#1}\def\tmpB{#2}}
\def\headandtail#1{\expandafter\mytmpfn #1\end}

\headandtail{dream}
\expandafter\uppercase\expandafter{\tmpA} (\tmpB)
```

D (ream)

일찍부터 이것이 매우 불편하여, LATEX에서 \MakeUppercase를 정의해두고 있습니다. \expandafter를 자동으로 해주는 것이라고 생각하면 됩니다.

```
\def\mytmpfn#1#2\end{\def\tmpA{#1}\def\tmpB{#2}}
\def\headandtail#1{\expandafter\mytmpfn #1\end}

\headandtail{dream}
\MakeUppercase{\tmpA} (\tmpB)
```

D (ream)

그런데, 또 생각해보면 \MakeUppercase는 하나의 인자를 취하여 그것을 확장하도록 정의되어 있으므로, 위와 같이 미리 분리해서 넘길 것 없이 그냥 한 개의 인자만 전달되도록 하면 될 것입니다. (\uppercase는 이것이 되지 않습니다.) 다음 두 결과를 비교하여 봅시다.

```
\MakeUppercase dream \\
\MakeUppercase{dream}
```

Dream
DREAM

\MakeUppercase를 쓰면서도 \expandafter가 필요할 수 있습니다. 다음 두 결과를 비교하여 봅시다.

```
\def\tmp{dream}
\expandafter \MakeUppercase \tmp \\
\MakeUppercase \tmp
```

```
Dream
DREAM
```

expl3의 방법 다른 분이 제출한 답안에 첫 글자만 분리해내어 처리하는 방법이 적용되어 있습니다.
첫 글자만 분리하는 방법으로서,

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { dream }
[ \tl_head:N \l_tmpa_tl ] ( \tl_tail:N \l_tmpa_tl )
\ExplSyntaxOff
```

```
[d](ream)
```

`\tl_head:N`과 `\tl_head:n`의 차이에 대하여 생각해보세요.

```
\tl_head:n { dream }
\tl_head:N \l_tmpa_tl
\tl_head:n { \l_tmpa_tl }
```

그러므로 첫 글자에 대해서만 조작을 가하려면

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { dream }
\MakeUppercase { \tl_head:N \l_tmpa_tl } ~(\tl_tail:N \l_tmpa_tl)
\ExplSyntaxOff
```

```
D (ream)
```

이것으로 좋다는 것을 알 수 있습니다. `\MakeUppercase`는 인자를 먼저 확장하도록 되어 있다는 점을 지적하였습니다.
한편, expl3에는 대문자나 소문자로 변환하는 함수가 미리 준비되어 있습니다.
`\text_uppercase:n`, `\text_lowercase:n`, `\text_titlecase:n`.
이 세 명령은 이름에는 `n`이라고 되어 있지만 기본적으로 인자를 먼저 확장하는 특별한 함수들입니다. 그렇기 때문에 `o`, `x` 등의 확장형이 따로 없습니다.[1] 이밖에 `\text_titlecase_first:n`이라는 것이 있지만 이것은 특별한 유럽어를 위한 것이므로 일단 논외로 하겠습니다.

```
\ExplSyntaxOn
Hello,~\text_titlecase:n { dongsu } \\
\tl_set:Nn \l_tmpa_tl { cheolsu } \\
Hello,~\text_titlecase:n { \l_tmpa_tl }
\ExplSyntaxOff
```

```
Hello, Dongsu

Hello, Cheolsu
```

[1] 이 글이 처음 쓰일 당시에는 위에 열거한 명령들이 expl3에 도입되기 전이었기 때문에 `\tl_upper_case:n`, `\tl_lower_case:n`, `\tl_mixed_case:n`으로 설명하였다. 2019년 11월부터는 이 명령 대신 `\text_...` 명령을 쓰도록 한다. 다만 2020년 현재 이전 버전과의 호환을 위하여 `tl`류 명령도 동작하기는 하지만 interface3 문서에는 빠져 있다.

이 예에서 보듯이 `\text_...` 명령의 인자형이 `:n`이지만 인자로 문자열 또는 매크로 어느 쪽이든 원하는 결과를 얻습니다. 그 이유는 이 명령이 인자에 대하여 `\text_expand`라는 함수를 한 번 실행하기 때문입니다.

str_함수 실제 case 관련 함수는 `\char_uppercase:N`과 `\str_uppercase:n`이 더 있습니다. `char_` 종류의 명령은 expl3 내부적으로 사용되는 경우가 많으므로 크게 신경쓸 것이 없고, `\str_uppercase:n`이 문제인데, 이것은 주로 매크로나 함수 이름의 case에 신경써야 하는 프로그래머를 위한 명령입니다. 예컨대,

```
\ExplSyntaxOn
\str_uppercase:n { \tmp }
\ExplSyntaxOff
```

```
\TMP
```

이와 같이 매크로 이름의 케이스를 바꿀 수 있습니다. 또한 두 문자열을 비교하려 할 때 case를 무시하고 비교할 수 있도록 하는 `\str_foldcase:n` 같은 것도 역시 프로그래밍을 위해서 필요한 함수입니다.
이런 상황이 아니라면 `\str_...` 역시 지금 당장은 크게 신경쓸 것이 없습니다. 지금의 예와 같이 사용자 문자열의 케이스를 문제삼을 적에는 `\text_<upper|lower|title>case:n`이 가장 적당합니다.

문제

문자열 인자를 받아서 각 글자마다 `\fbox`를 치는 명령 `\foobox`를 작성하여라.

입력: `\foobox{Korea}`
출력: K o r e a

1 new, set, map, use

expl3의 리스트 자료형 tl, clist, seq에 대하여,

`new` 새로운 변수를 선언한다. 단, 스크래치 변수라면 `new`가 필요없다.

`set` 변수에 새로운 값을 할당한다. (전역 변수에 대해서는 `gset`.)

`map` 리스트 각 항목(item)에 대하여 순차적으로 어떤 작용(function)을 적용한다.

`use` 리스트 변수를 확장(expand)하여 입력 스트림에 남긴다.

참고 1. 일부 자료형(특히 tl)은 `\tl_new:N`가 없어도 `\tl_set:Nn`할 수 있다. 그러나 `\tl_set:Nn`이 아닌 다른 tl 조작함수를 new하지 않고 부를 수는 없으므로 tmpa, tmpb가 아니고 사용자 변수를 만들어 쓴다면 new하는 것이 안전하다.

참고 2. `\tl_use:N \l_tmpa_tl`은 이것이 입력 스트림에 남겨지는 경우라면 `\l_tmpa_tl`과 결과가 동일하다. 그러나 차이라면 `\tl_use:N`이 쓰이면 대상 tl을 한 번(o) 확장한다는 것이다. 지금은 이 둘이 큰 차이가 없는 상황만을 주로 다루게 된다.

map에는 두 가지 방법이 있다. 하나는 inline이고 다른 하나는 function을 부르는 방식이다. 단순히 `\fbox`를 부르는 경우만을 예로 들면 다음과 같다.
inline mapping:

```
\tl_new:N \l_my_tl
\tl_set:Nn \l_my_tl { We shall overcome }
\tl_map_inline:Nn \l_my_tl
{
    \fbox { #1 }
}
\tl_use:N \l_my_tl
```

function mapping:

```
\cs_new:Npn \myfbox_fn:n #1
{
    \fbox { #1 }
}

\tl_new:N \l_my_tl
\tl_set:Nn \l_my_tl { We shall overcome }
\tl_map_function:NN \l_my_tl \myfbox_fn:n
\tl_use:N \l_my_tl
```

inline mapping에서 주의할 점은 이 코드가 다른 함수의 정의 범위 안에 들어가 있다면 #1이 아니라 ##1이 되어야 한다는 것이다. expl3는 ###1이라는 토큰이 의미가 없고 세번째 #은 ##으로 표현하여야 하기 때문에, 우리는 #의 중첩을 두 번까지만 허용하는 것으로 보기로 하자. nesting이 필요하면 외부 함수를 정의하고 이것에게 실행을 맡기는 방식을 주로 쓰기로 한다.
function mapping의 경우는 인자를 한 개 받아들이는 mapping function을 미리 정의해두어야 한다.
주어진 문제는 따라서 다음과 같이 해결할 수 있다.

```
\ExplSyntaxOn
\NewDocumentCommand \foobox { m }
{
    \my_foobox:n { #1 }
}

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

\cs_new:Npn \my_foobox:n #1
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_map_function:NN \l_tmpa_tl \f_box_it:n
}
\ExplSyntaxOff

\foobox{Korea}
```

K o r e a

clist와 seq를 연습해보자.

```
\ExplSyntaxOn
\clist_new:N \l_my_clist
\clist_set:Nn \l_my_clist { abc, 123, ABC }
\clist_map_function:NN \l_my_clist \myfbox_fn:n
\par
\clist_use:Nn \l_my_clist {,~}
\ExplSyntaxOff
```

abc 123 ABC
abc, 123, ABC

clist는 항목 구분자가 쉼표로 통일되어 있지만 seq는 매우 유연하다. 그 대신 항목 구분자를 반드시 매번 밝혀야 한다.

```
\ExplSyntaxOn
\seq_new:N \l_my_seq
\seq_set_split:Nnn \l_my_seq {|} { abc|123|ABC }
\seq_map_function:NN \l_my_seq \myfbox_fn:n
\par
\seq_use:Nn \l_my_seq {,~}
\ExplSyntaxOff
```

abc 123 ABC
abc, 123, ABC

seq에서 set을 위하여 `\seq_set_split:Nnn`과 `\seq_set_from_clist:Nn`이 흔하게 쓰인다.
seq와 clist는 간단한 db나 stack처럼 활용하게 될 수 있다. 굳이 말하자면 seq/clist는 다른 언어의 리스트에 가깝고 tl은 스트링과 비슷하다. clist와 seq의 다른 활용 가능성은 이어지는 강좌에서 다루게 된다.

map과 꼬리재귀 `map` 함수는 expl3 언어의 가장 중요한 기능 중의 하나다. 예를 들어 주어진 문자열에서 각 문자마다 fbox를 치는 이 단순한 일은

```
\def\acmd#1{\expandafter\aacmd#1\end}
\def\aacmd#1{\ifx#1\end\let\next\relax
    \else \fbox{#1}\let\next\aacmd\fi\next}
```

이렇게 정의하는 것이 plainTEX스러운 방법일 것이다. 또는 우리가 즐겨 사용하는

```
\def\fifo#1{\ifx#1\ofif\ofif\fi\process#1\fifo}
\def\ofif#1\fifo{\fi}
\def\process#1{\fbox{#1}}
\def\acmd#1{\expandafter\fifo#1\ofif}
```

fifo ... ofif 기법을 써도 마찬가지인데, 이 두 방법은 모두 이른바 "TEX의 꼬리재귀"를 적용하고 있다. 실상 리스트 매핑이라는 것은 꼬리재귀를 아주 쉽게 사용하도록 해둔 것이다.
expl3에서는 이른바 "꼬리재귀"를 직접 적용할 수 없는가? 물론 가능하다. 이것을 plainTEX에 비하여 더 안전하게 수행할 수 있도록 *quark* 데이터타입이 정의되어 있다. 당분간 이 자료형을 활용하게 될 일은 없을 것이지만 여기에서 재미삼아 (low-level expl3의) 예를 들어두고 간다.

```
\ExplSyntaxOn
\NewDocumentCommand \test { m }
{
    \test_fn:n #1 \q_recursion_tail \q_recursion_stop
}

\cs_new:Npn \test_fn:n #1
{
    \quark_if_recursion_tail_stop:n { #1 }
    \myfbox_fn:n { #1 }
    \test_fn:n
}
\ExplSyntaxOff

\test{abcde}
```

a b c d e

tl 관련 기본 함수 다음 함수가 자주 사용할 법한 것이다.

- `\tl_clear:N` 현재 tl 안의 내용을 모두 지운다. 이것은 tl 자체를 삭제하는 것과는 다르다.
- `\tl_put_right:Nn` n 인자의 내용을 현재의 tl의 뒤(오른쪽)에 붙여넣는다. 반면 `\tl_put_left:Nn`은 앞(왼쪽)에 붙인다.

- `\tl_set_eq:NN  \let\A\B`와 거의 동일하다. 두 개의 tl을 일치시킨다.
- `\tl_count:N` tl의 토큰 개수를 반환한다.
- `\tl_reverse:N` 현재 tl의 토큰을 역순으로 배열한다.

2 정수에 관한 기초 지식

수(numbers)를 본격적으로 다루기 전에 학습의 진행을 위해서 정수(*int*)에 대하여 다음 몇 가지를 알아두자.

new 정수형 변수는 사용 전에 반드시 new 선언을 하여야 한다. 단, tmpa, tmpb라는 스크래치 변수는 이미 정의되어 있다.

set `\int_set:Nn` 명령의 n인자 부분은 `\int_eval:n`으로 확장되어 적용된다.

integer expression 정수 표현식은 `+, -, *, /`과 괄호(`()`)로 이루어진다. 누승(지수)이나 계승(팩토리얼)은 정수와 관련 있음에도 불구하고 fp 자료형에서 담당한다. 참고로 `/` 연산은 (plainTEX의 `\divide`와 달리) truncate가 아니라 round이다. `\int_eval:n { 7/4 }`의 결과는 2. 같은 연산을 `\divide`로 하면,

```
\newcount\tmpcnt\tmpcnt=7
\divide\tmpcnt by4 \the\tmpcnt
```

결과는 1. trunc와 modulo를 위해서는 따로 함수가 준비되어 있다.

compare 정수 비교식은 `>`, `<`, `=` (`==`), `!=`, `>=`, `<=`를 비교 연산자로 하고 bool 값을 반환한다. 이것은 `\int_compare:nTF`의 첫 인자로만 쓰인다. 이 함수 `\int_compare:nTF`는 사실상 expl3의 정수형 if-문이므로 그 사용례를 숙지해두어야 한다.

use tl과는 달리 `\l_tmpa_int`만을 입력 스트림에 남기면 에러가 발생한다. 정수형은 반드시 `\int_use:N \l_tmpa_int` 꼴로 쓰거나 `\int_eval:n`한 다음에야 tl에 할당될 수 있다.

수에 대하여 따로 공부하기 전에 당장 필요한 것은 다음 몇 가지 명령이다.

- `\int_zero:N` 주어지는 변수의 값을 0으로 만든다. `\int_set:Nn \l_tmpa_int {0}`와 동일. 편의를 위한 `\int_zero_new:N`이라는 것도 있다.
- `\int_incr:N` 주어지는 변수의 값을 1 증가시킨다. `\int_add:Nn \l_tmpa_int {1}`과 동일.
- `\int_compare:nTF` n에는 정수 비교식이 온다.
- `\int_use:N`
- `\int_if_odd:nTF`, `\int_if_even:nTF`

연습문제

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

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.

---

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

문제

주어지는 문자열을 3개 단위로 끊어서 다음 출력예와 같이 배열하여라. 마지막 항목은 3개가 되지 않을 수 있으며 스페이스는 무시한다.

입력: `\scmd{abcdefg hijklmn}`
출력: abc def
ghi jkl
mn

`\tl_map`할 적에 스페이스(10)는 하나의 토큰(next token)으로 치지 않는다. 스페이스를 다루는 방법에 대하여는 지금 다루지 않는다.

```
\ExplSyntaxOn
\NewDocumentCommand \test { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_map_inline:Nn \l_tmpa_tl
    {
        \fbox { ##1 }
    }
    (\tl_count:N \l_tmpa_tl)
}
\ExplSyntaxOff
\test{a bc def gh}
```

a b c d e f g h (8)

주어진 문제는 글자 수를 세는 것이 관건이다.

인덱스 카운터를 이용하자 `\tl_map`으로 주어지는 토큰을 하나씩 처리하는 상황을 생각한다. 이 때 임의의 정수 `\l_tmpa_int`를 생각하자.

(1) 첫 번째 아이템이 들어오면 `\l_tmpa_int`를 1 증가시킨다.

(2) 첫 번째 아이템을 임시 tl `\l_tmpb_tl`에 적립(`\tl_put_right:Nn`)한다.

(3) 이 과정을 반복해가다가, `\l_tmpa_int`가 3이 되면 카운터를 0으로 만들고 지금까지 적립된 `\l_tmpb_tl`을 출력(입력 스트림에 남김)한 다음 `\l_tmpb_tl`을 비운다(clear).

```
\ExplSyntaxOn
\NewDocumentCommand \test { m }
{
    \int_zero:N \l_tmpa_int
    \test_process:n { #1 }
}

\cs_new:Npn \test_process:n #1
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_map_inline:Nn \l_tmpa_tl
    {
        \int_incr:N \l_tmpa_int
        \tl_put_right:Nn \l_tmpb_tl { ##1 }
```

```
        \int_compare:nTF { \l_tmpa_int == 3 }
        {
            \tl_use:N \l_tmpb_tl \par
            \int_zero:N \l_tmpa_int
            \tl_clear:N \l_tmpb_tl
        }
        { }
    }
}
\ExplSyntaxOff
\test{abcdefg hijk}
```

abc
def
ghi

위의 예에서 `\int_compare:nTF`에서 F 부분은 아무 것도 하는 것이 없다. 이럴 경우에 F를 생략하고 해당 부분을 적지 않을 수 있다.
이 코드는 다 좋은데 마지막에 남는 두 글자가 (3을 이루지 못했기 때문에) 식자되지 않았다. 이를 처리하기 위해서 다음 코드를 추가한다.

```
\int_compare:nT { \tl_count:N \l_tmpb_tl != 0 }
{
    \tl_use:N \l_tmpb_tl \par
}
```

이제 마지막으로 출력 모양에 대하여 생각해본다. 위의 예에서 매번 `\l_tmpb_tl \par`를 하여 추려낸 것을 한 줄씩 찍었다. 이것을 어떤 tl에 다음과 같은 모양으로 저장하였다가

```
abc & def \tabularnewline
```

tabular 환경 안에서 확장해주면 될 것 같다.
이를 위하여 `\l_tabitem_int`라는 정수(카운터)와 `\l_tablines_tl`이라는 tl을 하나 마련하였다. 다음 코드를 보고 무슨 일이 일어난 것인지 잘 살펴보도록 하여라.

```
\ExplSyntaxOn
\tl_new:N \l_tablines_tl

\NewDocumentCommand \testa { m }
{
    \int_zero:N \l_tmpa_int
    \tl_clear:N \l_tablines_tl
    \int_zero_new:N \l_tabitem_int

    \testa_fn:n { #1 }

    \begin{tabular}{ll}
    \tl_use:N \l_tablines_tl
    \end{tabular}
}

\cs_new:Npn \testa_fn:n #1
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_map_inline:Nn \l_tmpa_tl
    {
```

```
        \int_incr:N \l_tmpa_int
        \tl_put_right:Nn \l_tmpb_tl { ##1 }

        \int_compare:nT { \l_tmpa_int == 3 }
        {
            \int_incr:N \l_tabitem_int
            \int_if_odd:nTF { \l_tabitem_int }
            {
                \tl_put_right:Nx \l_tablines_tl { \l_tmpb_tl }
                \tl_put_right:Nn \l_tablines_tl { \c_alignment_token }
            }
            {
                \tl_put_right:Nx \l_tablines_tl { \l_tmpb_tl }
                \tl_put_right:Nn \l_tablines_tl { \tabularnewline }
            }
            \int_zero:N \l_tmpa_int
            \tl_clear:N \l_tmpb_tl
        }
    }
    \int_compare:nT { \tl_count:N \l_tmpb_tl != 0 }
    {
        \int_incr:N \l_tabitem_int
        \int_if_odd:nTF { \l_tabitem_int }
        {
            \tl_put_right:Nx \l_tablines_tl { \l_tmpb_tl }
            \tl_put_right:Nn \l_tablines_tl { \c_alignment_token
             ↪  \tabularnewline }
        }
        {
            \tl_put_right:Nx \l_tablines_tl { \l_tmpb_tl }
            \tl_put_right:Nn \l_tablines_tl { \tabularnewline }
        }
    }
}
\ExplSyntaxOff
\testa{ABCD efghi jklmno p}
```

ABC	Def
ghi	jkl
mno	p

인덱스에 대하여 modulo 연산 위의 예에서는 인덱스가 3이 되면 이를 0으로 되돌리는 방법을 사용하였다. 이렇게 하지 않고 인덱스 카운터를 증가시키면서 그 값의 modulo(3) 연산 결과가 0인가를 검사하는 방법이 있다. 이에 대해서는 따로 코드를 보이지 않을 것이니 꼭 실제 해보기를 바란다. `\int_mod:nn`을 쓰면 간단히 된다.

regex로 해보자 expl3가 plainTEX이나 기존 LATEX에 비하여 가진 중대한 장점이 sorting과 regex 관련 함수를 제공한다는 점이다.
시나리오는 간단하다. 인자로 주어지는 문자열을 regex 조작하여 세 글자 이후마다 어떤 표지(예컨대 쉼표)를 붙인 다음에 이를 seq 또는 clist로 처리하는 것이다. 코드가 매우 세련되고 아름다워지는 장점이 있다. regex 함수들이 실행상 약간의 부하가 걸린다는 주장이 있지만 말단 사용자로서는 크게 신경쓸 일이 아니다.

[붙임 1] 정규식(regular expression)이 아닌 문자(열) 바꾸기라면 l3tl 함수인 `\tl_replace_once:Nnn` 이나 `\tl_replace_all:Nnn`을 쓰면 된다.

`sed`로 'abcdefghij'에 대하여 세 글자마다 쉼표를 붙이려면

```
$ sed 's/\(.\)\(.\)\(.\)/\1\2\3,/g' < file_in.txt
```

이와 같이 한다. expl3의 l3regex로 간단히 다음과 같이 할 수 있다.

```
\tl_set:Nn \l_tmpa_tl { abcdefgh ijk }
\regex_replace_all:nnN { (.)(.)(.) } { \1\2\3 , } \l_tmpa_tl
```

또는 더 간단하게

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { abcdefghijk }
\regex_replace_all:nnN { (.){3} } { \0 , } \l_tmpa_tl
\tl_use:N \l_tmpa_tl
\ExplSyntaxOff
```

```
abc,def,ghi,jk
```

이제 `\l_tmpa_tl`에 들어 있는 것은 쉼표로 분리된 토큰열이므로 clist를 사용할 수 있겠다. 물론 쉼표가 아닌 다른 표지를 넣을 수 있으며, 이 때는 seq로 처리하는 것이 어렵지 않다. 분리 표지를 |로 하고 seq로 같은 일을 하도록 하는 코드를 연습해보라.
tabular를 위해서 행하는 조작 부분을 직접 해보는 즐거움을 위해 남겨두고 여기서는 해당 항목을 출력한 후 개행하는 방식의 짧은 코드만 보이기로 한다. (항목만을 출력하고 개행하는 아래 코드는 사실 `\clist_map_...`하지 않아도 간단히 `\clist_use:Nn`으로 처리할 수 있다. 그러나 아래 코드를 그대로 남겨두는 이유는 이 부분에 tabular를 만들기 위한 조작을 넣어야 하기 때문이다.)

```
\ExplSyntaxOn
\NewDocumentCommand \testc { m }
{
    \testc_fn:n { #1 }
}

\cs_new:Npn \testc_fn:n #1
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \regex_replace_all:nnN { \s } { } \l_tmpa_tl
    \regex_replace_all:nnN { (.){3} } { \0, } \l_tmpa_tl
    \clist_set:No \l_tmpa_clist { \l_tmpa_tl }

    \clist_map_inline:Nn \l_tmpa_clist
    {
        ##1 \par
    }
}
\ExplSyntaxOff

\testc{abcd efgh ijk lmno}
```

```
abc
def
ghi
jkl
mno
```

`\regex_...` 명령이 두 번 쓰였는데 첫 번째 것은 space 문자를 제거하는 것이다. regex 명령에 들어오는 tl 안의 모든 문자가 보존되므로 이와 같이 하여 스페이스를 제거할 필요가 있다.

한편, `\clist_set:No`가 사용되었다. 다음 두 명령은 그 의미가 같다.

```
\clist_set:No \l_tmpa_clist { \l_tmpa_tl }
\clist_set:NV \l_tmpa_clist \l_tmpa_tl
```

요컨대 `\l_tmpa_tl`이라는 매크로 자체를 취하지 말고 그것을 확장하여 그 값(*value*)을 취하라는 것이다.

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.

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.

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.

1. 입력: `\acmd{5}{beautiful}\quad \acmd{5}{abc}`
   출력: beaut __abc
2. 입력: `\myslicing{Hello world}{3}{7}`
   출력: llowo
3. 입력: `\myitemswap{abcde}{2}{4}{\myresult}`
   출력: `\myresult` → adcbe

문제

다음 실행의 결과가 어떠할지 예측해보아라. 실제로 예상과 같은지 확인해보아라.

```
\ExplSyntaxOn
\tl_new:N \l_tmpc_tl

\tl_set:Nn \l_tmpa_tl { foo }
\tl_set:Nn \l_tmpb_tl { \l_tmpa_tl }
\tl_set:No \l_tmpc_tl { \l_tmpa_tl }

\tl_set:Nn \l_tmpa_tl { bar }

\tl_use:N \l_tmpb_tl
\tl_use:N \l_tmpc_tl
\ExplSyntaxOff
```

3 확장(expansion)의 기초

이 코드를 실행하면 다음과 같은 결과가 나온다.

```
\ExplSyntaxOn
\tl_use:N \l_tmpb_tl \par
\tl_use:N \l_tmpc_tl
\ExplSyntaxOff
```

```
bar
foo
```

이유는 `\l_tmpb_tl`에는 `{\l_tmpa_tl}`이 들어 있지만 `\l_tmpc_tl`에는 set 명령이 실행되는 시점에서 `\l_tmpa_tl`이 가지고 있던 값이 해동되어서 들어가 있기 때문이다. 추후 `\l_tmpa_tl` 값을 바꾼다면 이 매크로 자체를 가지고 있는 `\l_tmpb_tl`은 당연히 달라진 값을 식자하겠지만 `\l_tmpc_tl`은 그 영향을 받지 않는다.

expl3의 함수 인자형 지시자가 번거롭기만 하고 TMI가 아니냐는 의견이 있는데 전혀 그렇지 않다. expl3로 코딩하는 프로그래머는 자신이 사용하고 있는 매크로(변수)가 특정 시점에서 확장되어야 할지 그렇지 않은지를 항상 세심하게 유념하여야 한다. 또한 인자를 확장시키는 다양한 방법이 존재하는데, 이에 대해서 다음 기회에 더 자세히 다룬다.

다른 한 예를 들어보자. 앞서 tabular 안에 자신이 수집한 매크로를 바로 넣는 방법, 즉

```
\begin{tabular}{ll}
\tl_use:N \l_tablines_tl
\end{tabular}
```

와 같은 코드를 생각해보자. expl3 이전이면 이런 일을 어떻게 하였을까?

```
\makeatletter
\protected\def\tnewlinehline{\tabularnewline\hline}
\def\@tablines{abc & def \tnewlinehline }
\protected@edef\@tablines{\@tablines LMN & OPQ \tnewlinehline }
\protected@edef\@tablines{\@tablines rst & uvw \tnewlinehline }

\begin{tabular}{l|l}
```

```
\hline
\@tablines
\end{tabular}
\makeatother
```

abc	def
LMN	OPQ
rst	uvw

이것이 전통적으로 사용되어온 기법이다. (혹시 쓰일 데가 있을지 모르니 알아두는 것도 좋다.) 이 코드에서 주의할 것은 두 가지인데, 하나는 `\hline`이 fragile한 명령이라서 이것을 `\@tablines`에 직접 집어넣으면 에러가 뜬다는 것이다. 그래서 `\protected\def`으로 한 번 묶어주었다.
그리고 `\protected@edef\@tablines{\@tablines...` 부분을 유의해서 보아야 한다. `\@tablines`을 확장하여 기존에 가지고 있던 값을 얻고 그 뒤에 새로운 것을 추가하였다.
이 코드에 대응하는 expl3는 다음과 같다.

```
\ExplSyntaxOn
\tl_set:Nn \l_tablines_tl { abc \c_alignment_token def \tabularnewline \hline
 ↪ }
\tl_put_right:Nn \l_tablines_tl { LMN \c_alignment_token OPQ \tabularnewline
 ↪ \hline }
\tl_put_right:Nn \l_tablines_tl { rst \c_alignment_token uvw \tabularnewline
 ↪ \hline }

\begin{tabular}{l|l}
\hline
\tl_use:N \l_tablines_tl
\end{tabular}
\ExplSyntaxOff
```

abc	def
LMN	OPQ
rst	uvw

번거로운 `\protected@edef` 대신 `\tl_put_right:Nn`을 쓰면 된다는 것을 알겠다. `\hline`은 특히 확장되지 않고 들어가도록 (즉 `n` 인자로) 조심하는 것이 좋다.

`\use:c`에 관하여 `\tl_set:Nn` 대신 `\tl_set:cn`을 쓰면 다음과 같은 일을 할 수 있다.

```
\ExplSyntaxOn
\int_zero:N \l_tmpa_int
\int_incr:N \l_tmpa_int
\tl_set:cn { mycmd \int_to_Alph:n { \l_tmpa_int } } { result~=~1 }
\int_incr:N \l_tmpa_int
\tl_set:cn { mycmd \int_to_Alph:n { \l_tmpa_int } } { result~=~2 }
\ExplSyntaxOff
\mycmdA, \mycmdB
```

result = 1, result = 2

plainTEX에서 다음과 같이 하던 것에 해당한다.

```
\newcount\mycnt
\mycnt=1
\expandafter\def\csname mycmd\romannumeral\mycnt\endcsname
{ result 1 }
\advance\mycnt by1
\expandafter\def\csname mycmd\romannumeral\mycnt\endcsname
{ result 2 }

\mycmdi, \mycmdii
```

result 1 , result 2

연습문제

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

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

힌트: 이 문제는 앞서 그 해법이 이미 다루어졌다. 다시 연습문제로 내는 이유는 만약 입력되는 문자열의 개수가 3의 배수이면 마지막에 쉼표가 붙을 수 있는데 이것이 붙지 않도록 하라는 것이다.

문제

새로운 명령 \newcmd는 다음과 같은 형식으로 실행한다.
\newcmd{abc, de, fgh, i, jkl}
쉼표로 분리된 각 항목을 enumerate으로 배열하여라.

입력: \newcmd{abc, de, fgh, i, jkl}
출력:

1. abc
2. de
3. fgh
4. i
5. jkl

이 문제는 KTUG QnA:236820에서 질문과 답변이 이루어진 것이다. 해당 글타래에 expl3를 사용하는 답변이 없다. 지금까지 학습한 것으로 문제를 해결할 수 있을 것이므로 이를 작성하여 보아라.

4 리스트의 소팅

여기서 배우고자 하는 것은 sorting이다. 숫자로 된 리스트의 소팅은 다음과 같이 한다. (이하 clist만을 다루는데 seq에 대해서도 마찬가지로 동작한다.)

```
\ExplSyntaxOn
\clist_set:Nn \l_tmpa_clist { 29, 8, -3, 6, 12 }
\clist_sort:Nn \l_tmpa_clist
{
    \int_compare:nTF { #1 > #2 }
    { \sort_return_swapped: }
    { \sort_return_same: }
}
\clist_use:Nn \l_tmpa_clist {,~}
\ExplSyntaxOff
```

-3, 6, 8, 12, 29

\..._sort:Nn 함수는 그 정의에서 #1과 #2를 비교한다. 이것이 다른 함수 정의 안에서 사용될 적에는 ##1과 ##2가 되는 것에 주의하여야 한다.
문자열의 소팅은 어떻게 할 것인가? 다음처럼 하는 것이 한 가지 방법이다.

```
\ExplSyntaxOn
\clist_set:Nn \l_tmpa_clist { dog, tiger, lion, sheep, cat, mouse }
\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
```

cat, dog, lion, mouse, sheep, tiger

여기서 "Do not use"라고 하는 `\tex_strcmp:D`를 사용하였는데 만약 이를 사용하지 않으려면 좀더 복잡한 코딩이 필요하기 때문에 간편하게 이를 빌어 쓰기로 하였다. (참고로 LuaTEX에서는 `l3kernel.strcmp`라는 함수를 사용할 수 있다.)
어떤 clist (book, house, building, beer, cat)를 사전순(alphabetically)으로 소트할 것이 아니라 문자열 길이를 기준으로 소팅하고 싶다면 어떻게 해야 할까? 연습문제 삼아 풀어보기 바란다. 결과는 다음과 같이 나온다. (만약 길이가 같으면 알파벳순으로 정렬하도록 2중 조건을 적용하였다.)

```
\ExplSyntaxOn
\clist_use:Nn \l_tmpa_clist { ,~ }
\ExplSyntaxOff
```

cat, beer, book, house, building

tl의 각 항목도 정렬할 수 있다. 즉 `\tl_sort:Nn`를 이용할 수 있다.
소팅은 요긴하지만 LATEX 프로그래밍의 관점에서는 아주 예외적인 상황에서 필요하다. 실제 문헌목록이나 인덱스 등에서의 소팅은 makeindex, bibTEX 프로그램이 하는 것이므로 이 경우와는 같지 않다.

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.

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

1. 입력: `\newcmd{this is just a test}`
   출력: `\resi` → thi, `\resiv` → tat
2. 입력: `\testcmd{abaracadabra}`
   출력: a = 6, b = 2, c = 1, d = 1, r = 2

문제

1부터 입력받은 정수(자연수)까지의 합을 출력하여라.

입력: `\summation{10}`
출력: $\sum_{k=1}^{10} = 55$

n까지의 합을 구하는 것은

$$\sum_{k=1}^{n} = \frac{n(n+1)}{2}$$

로 간단히 처리할 수 있다.

```
\ExplSyntaxOn
\NewDocumentCommand \simplesum { m }
{
    \int_eval:n { #1 * (#1+1) / 2 }
}
\ExplSyntaxOff

\simplesum{10}
```

55

나누기가 있는데 정수형으로 반환해도 괜찮을까? 물론 괜찮다. 이유는 $n(n+1)$ 둘 중 하나는 반드시 짝수일 것이기 때문이다. 만약 정수 아닌 결과가 예상되는 경우이고 반드시 정수형으로 값을 내어야 하며 나눗셈이 포함된 식이라면 `\int_eval:n`의 결과가 정수이기는 하겠지만 반올림한 결과일 것이라는 점을 염두에 두어야 할 때가 있을 수 있다.

5 함수와 프로시저

Pascal에서는 function과 procedure를 구별한다. `return`이 있는, 즉 어떤 값을 반환하는 것은 function이고 반환값 없이 일정한 처리만을 하는 것이 procedure이다.
expl3의 함수(cs)들은 **근본적으로** procedure이다. 그러나 그 "처리"가 입력 스트림에 무엇인가를 남기는 것이라면 그것을 반환값처럼 활용할 수는 있다. 어떤 사용자 정의 함수가 반환값을 가지고 있다고 하더라도 그것이 다른 함수의 인자로 들어가서 확장될 수 있느냐는 또 다른 문제이다. 그래서 expl3 코딩에서는 가능하면 변수 조작을 통해서 문제를 해결하는 것이 복잡한 확장 문제를 피해가기 더 나을 때가 있다.

```
\ExplSyntaxOn
\int_new:N \g_totalsum_int

\NewDocumentCommand \simplesum { m }
{
    \int_gzero:N \g_totalsum_int
    \calc_sum:n { #1 }
    \int_use:N \g_totalsum_int
}
```

```
\cs_new:Npn \calc_sum:n #1
{
    \int_gset:Nn \g_totalsum_int { #1 * ( #1 + 1 ) /2 }
}
\ExplSyntaxOff

\simplesum{10}
```

55

이 예시에서 \calc_sum:n은 그 자체로 아무 것도 반환하지 않는다. 그 대신 \g_totalsum_int라는 전역 변수의 값을 조작하는 "처리"를 행하고 있다.

확장 (2): 인자의 한 번 확장

이 둘의 차이를 조금 더 살펴보자.
다음 보기를 잘 보아라. 여기서 정의된 \fn_sum:n은 "함수"처럼 정의한 것이다. 그러므로 일정한 값을 "반환"할 것이고, 따라서 \fn_sum:n {\fn_sum:n { 5 } }라고 명령하면 \fn_sum:n{15}의 효과가 나타나야 한다.

```
\ExplSyntaxOn
\cs_new:Npn \fn_sum:n #1
{
    \int_eval:n { #1 * ( #1 + 1 ) / 2 }
}

\exp_args:No \fn_sum:n { \fn_sum:n { 5 } } ~=~ \fn_sum:n { 15 }
\ExplSyntaxOff
```

120 = 120

그런데, 잘 생각해보면 \fn_sum:n은 숫자가 인자로 들어올 것을 예상하고 있다. 그러므로 인자에 들어가는 것을 확장해주지 않으면 안 된다. 이것은

```
\expandafter \fn_sum:n \expandafter { \fn_sum:n { 5 } }
```

와 같이 해야 할 것인데, expl3에는 인자를 확장하는 손쉬운 방법이 많이 제공된다. 인자를 확장하는 \exp_args:라는 (인자형에 따라 여러 가지 variant가 있는) 함수를 알아두자. 이를 이용하여 여러 번 \expandafter를 붙여야 하는 불편을 거의 피해갈 수 있다.
\exp_args:No에서 N 하나와 o 하나를 지정하였는데 여기서 확장되는 것은 두 번째 인자 o이다. o는 중괄호로 전달되는 그 안의 것을 "한 번" (once) 확장한다는 의미이다. 따라서 다음과 같이 쓸 수 있다. (이 자리에서는 :Nx로 해도 같은 결과이다. o와 x의 차이에 대해서는 다음에 토론할 기회가 있을 것이다.)

```
\exp_args:No \fn_sum:n { \fn_sum:n { 5 } }
```

그러면 \tl_set:No \l_a_tl { a }라는 형식의 명령은

```
\exp_args:NNo \tl_set:Nn \l_a_tl { a }
```

이것과 동일한 의미임을 알수 있겠다. 항상 n이 기본적인 형태이고 o가 그 확장형임을 기억해두자.
코딩의 가독성과 간결성을 위해서 \exp_args:는 남용하지 않는 것이 좋다. 그러나 확장이 문제로 될 때는 당연히 써야 한다. 확장 문제는 앞으로 지속적으로 새로운 형태를 배워가게 된다.

붙임 1: xparse의 \NewDocumentCommand는 기본적으로 protected 명령을 만든다. 그렇기 때문에 이렇게 정의된 명령을 확장하려면 더 깊은 확장 단계를 지시해야 한다. 즉시 확장 가능한 명령을 만드는 \NewExpandableDocumentCommand가 있지만 뭔가 길고 불편하다.

붙임 2: expl3를 쓰는 한, 인자형에 대해서 규칙을 지켜주는 것이 좋다. 나중에 틀림없이 헷갈리는 날이 온다. 예를 들어 `\int_mod:nn \l_tmpa_int 3`과 같은 코드는 비록 실행이 된다고 하더라도 혼란스럽다. 왜냐하면 n은 "중괄호 범위"로 전달되는 것으로 약속했기 때문이다. `\my_func:n \l_tmpa_tl`이 처음에는 좋은 듯하지만 이제 배운 "확장"이 문제가 되면 이것을 o확장해야 할지 V확장해야 할지 틀림없이 헷갈리는 날이 오는 것이다. 번거롭더라도 o확장을 의도한다면 `{ \l_tmpa_int } {3}`으로 적도록 습관을 들여야 한다. 굳이 위와 같이 간결하게 쓰고 싶다면, `\int_mod:VN`과 같은 형식으로 써두어야 나중에 헷갈리지 않는다. (그런 함수가 제공되지 않는다는 것은 일단 논외로 하고.)

6 반복문

이것만으로는 공부가 되기에 부족하므로, 다음과 같은 방법으로 합을 구해보고자 한다.

```
>>> def sum(n):
        s = 0
        for i in range(1,n+1):
                s += i
        return s

>>> sum(10)
55
>>>
```

expl3의 정수형(int)은 부호있는 32비트 정수(signed (long)int)이다. (8비트 CPU 시절에는 이것이 long이었다.) 표현범위는 -2^{31} (-2147483648) 부터 $2^{31}-1$ (2147483647)까지이다. `\int_eval:n`의 결과가 이 범위를 벗어나면 `Arithmetic overflow`라는 에러를 보이고 동작을 멈춘다.

붙임 3: int는 TEX의 count를 ε-TEX이 확장한 것이다. ε-TEX의 `\numexpr`가 정수 표현식의 출발점이다. 예를 들면 `\int_eval:n { a/b }`가 truncate가 아니라 round인 것도 ε-TEX으로부터 시작된 것이다. expl3의 정수는 ε-TEX 엔진에 전적으로 의존한다.

붙임 4: 32비트를 넘는 큰 수에 대해서 다루려면 xint 엔진을 이용할 수 있는데 이에 대해서 다음 강좌에서 간단히 취급할 생각이다.

6.1 for loop

for-loop는 카운터 인덱스가 있는 반복문이다.
표준적인 for-loop 문의 형식

```
for i := a to b step c
...
endfor
```

에서 i는 인덱스 카운터이고 a는 first, b는 last, c는 step이다. expl3에는 다음 함수들이 이 역할을 한다.

- `\int_step_function:nN` 1부터 주어지는 수까지 N을 반복. start는 1이고 step도 1. last가 첫 번째 인자 n이다. 반복할 함수는 N으로 주어진다. 함수가 별도로 정의되지 않고 inline으로 처리할 때 `\int_step_inline:nn`을 쓴다. 인덱스 카운터가 #1(다른 함수 정의 내부의 inline 함수라면 ##1)이다.
- `\int_step_function:nnN` 시작 숫자가 1일 아닐 때. 첫 번째 n이 start, 두 번째 n이 last이다. 마찬가지로 인덱스는 #1이며 inline 함수는 `\int_step_inline:nnn` 꼴로 쓴다.
- `\int_step_function:nnnN` step이 1이 아닐 때. 차례로 start, step, last 순이다. (start, last, step이 아니므로 순서에 주의). 마찬가지로 `\int_step_inline:nnnn`이 있다.

간단한 예를 들어두자.

```
\ExplSyntaxOn
\int_step_inline:nn { 5 } { #1 \quad } \par
\int_step_inline:nnn { 2 } { 6 } { #1 \quad } \par
\int_step_inline:nnnn { 3 } { 2 } { 11 } { #1 \quad }
\ExplSyntaxOff
```

```
1  2  3  4  5
2  3  4  5  6
3  5  7  9  11
```

다른 범용언어의 for 문과 비교하자면, 숫자를 인덱스로 하는 for-문은 `\int_step_...`과 유사하고 리스트를 인덱스로 하는 for-문은 이미 배운 바 `\..._map_...`과 비슷하다. 단, `\int_step_...` 함수들은 루프 중에 중단할 수 없다. (루프의 탈출이 중요하다면 `step` 함수를 쓰지 말고 `map` 함수를 사용하도록 하라.) 이를 이용하여 앞서 예시한 알고리즘을 expl3로 쓰면,

```
\ExplSyntaxOn
\int_new:N \g_sum_int

\NewDocumentCommand \suma { m }
{
    \int_gzero:N \g_sum_int
    \summation_fn:n { #1 }
    \int_use:N \g_sum_int
}

\cs_new:Npn \summation_fn:n #1
{
    \int_step_inline:nn { #1 }
    {
        \int_gadd:Nn \g_sum_int { ##1 }
    }
}
\ExplSyntaxOff

\suma{100}
```

```
5050
```

6.2 while, until

for를 쓰지 않고 while 반복문으로 같은 일을 할 수 있다.

```
def suma(n):
    a=0
    s=0
    while a<n:
        a+=1
        s+=a
    return s
```

expl3에서 while형 반복문은 다음 네 가지 형태가 있다.

(1) `..._while_do:nn`

(2) `..._do_while:nn`

(3) `..._until_do:nn`

(4) `..._do_until:nn`

의미는 직관적으로 이해가 될 것이다. 이 함수의 첫 인자 n은 실은 boolean 값을 반환하는 비교연산식으로 이루어진다. 그러므로 이 함수 형식의 원형은

```
\bool_while_do:nn { <bool expr> }
{ ... }
```

이러한 것이다. `<bool expr>` 부분에 예컨대 하나의 boolean 변수가 온다고 가정하자.

```
\ExplSyntaxOn
\bool_set_true:N \l_tmpa_bool
\int_zero:N \l_tmpa_int
\bool_while_do:nn { \l_tmpa_bool }
{
    \int_incr:N \l_tmpa_int
    \int_compare:nTF { \l_tmpa_int > 10 }
    {
        \bool_set_false:N \l_tmpa_bool
    }
    {
        a\int_use:N \l_tmpa_int
        \quad
        \bool_set_true:N \l_tmpa_bool
    }
}
\ExplSyntaxOff
```

a1 a2 a3 a4 a5 a6 a7 a8 a9 a10

bool 연산식과 자료형 bool 연산식은 다음 연산자로 이루어진다. && (and), || (or), ! (not)과 괄호(()). 이제 정수형 데이터에 대하여 while을 어떻게 적용할 것인가를 생각하자. `\l_tmpa_int` 값이 10보다 작으면 true, 그렇지 않으면 false가 되도록 하고 inline function을 true 조건의 while 반복문을 걸어보기로 한다.

```
\ExplSyntaxOn
\int_zero:N \l_tmpa_int

\bool_while_do:nn { \int_compare_p:n { \l_tmpa_int < 10 } }
{
    \int_incr:N \l_tmpa_int
    \int_to_Alph:n { \l_tmpa_int }
}
\ExplSyntaxOff
```

ABCDEFGHIJ

`..._p`가 붙는 함수는 boolean 값을 반환한다. 앞서 배운 `\int_compare:nTF`는 사실

```
\bool_if:nTF { \int_compare_p:n { ... } }
{ <T> } { <F> }
```

를 축약한 꼴이다. 이런 종류의 함수로 예컨대 `\tl_if_eq_p:NN`이 있는데 이것은 boolean값을 반환하므로

```
\bool_if:nTF { \tl_if_eq_p:NN <N> <N> } { <T> } { <F> }
```

라고 쓸 수 있는데 이를 간단히 `\tl_if_eq:NNTF`로 쓸 수 있는 것이다.
이제 `\bool_while_do:nn { \int_compare_p:n { ... } }` 이라고 써야 할 것을 간단히 `\int_while_do:nn`
으로 쓸 수 있음을 알게 되었다.
그리하여, 각 자료형에 대하여 while 함수가 여러 개 정의된 것처럼 보인다. 실은 `\bool_while_do:nn`의 첫 인자로 각 자료형의 bool 반환 함수를 쓰는 것을 줄여쓰게 한 것인데, 이들을 이런 식으로 설명하는 이유는 두 가지 이상의 조건을 and 또는 or 연산하려면 `\bool_while_do:nn`을 쓸 수 밖에 없기 때문이다. 즉, "임의의 수가 10보다 크다면 TF하라"는 명령은

```
\int_compare:nTF { \l_tmpa_int > 10 } { <T> } { <F> }
```

로 쓸 수 있지만, a가 10보다 크고 b가 100보다 작으면이라는 조건은

```
\bool_if:nTF
{
  \int_compare_p:n { \l_tmpa_int > 10 }
    &&
  \int_compare_p:n { \l_tmpb_int < 100 }
}
{ <T> } { <F> }
```

으로 써야 하는데 이에 대한 감각을 익혀두라는 의미이다. (이 정수 비교식은

```
\int_compare:n { 10 < \l_tmpa_int < 100 }
```

으로 줄여쓸 수 있지만 역시 위에 보인 것의 축약형이라고 이해해야 한다.)

아무튼 각 자료형 별로 while do 형식의 함수를 대강 보면

```
\dim_while_do:nn    \dim_do_while:nn    \dim_until_do:nn    \dim_do_until:nn
\fp_while_do:nn     \fp_do_while:nn     \fp_until_do:nn     \fp_do_until:nn
\int_while_do:nn    \int_do_while:nn    \int_until_do:nn    \int_do_until:nn
```

등이 있는 것이다.
bool 자료형(l3bool)이 변수로서 사용될 수 있다. 이 경우에는

```
\bool_while_do:Nn
```

을 쓸 수 있는데 N 위치에 임의의 boolean 변수(`\l_tmpa_bool`)가 올 수 있다. 참고로 bool 자료형에서 값을 할당할 적에는

```
\bool_set_true:N,     \bool_set_false:N,     \bool_set_inverse:N
```

이 세 가지 형식을 사용한다는 것을 알아두자.

do while과 while do while과 until의 차이는 설명할 필요 없다. 조건이 충족되면 inline 함수를 반복하는 것이 while이고 조건이 충족되면 반복을 멈추는 것이 until이다.
do while과 while do의 차이는 inline 반복 함수를 수행한 다음 조건을 검사할 것이냐 조건 검사를 먼저 할 것이냐의 차이이다.

```
\ExplSyntaxOn
\int_zero:N \l_tmpa_int
\int_while_do:nn { \l_tmpa_int < 0 } { X } \par
\int_do_while:nn { \l_tmpa_int < 0 } { Y } \par
\ExplSyntaxOff
```

```
Y
```

이 예에서 while do는 실행되지 않지만 do while은 최소한 한 번은 실행된다.

이제 원래의 문제를 while을 이용하여 해결해보자.

```
\ExplSyntaxOn
\int_new:N \l_sum_int
\NewDocumentCommand \sumw { m }
{
    \int_zero:N \l_tmpa_int
    \int_zero:N \l_sum_int
    \int_do_while:nn { \l_tmpa_int < #1 }
    {
        \int_incr:N \l_tmpa_int
        \int_add:Nn \l_sum_int { \l_tmpa_int }
    }
    \int_use:N \l_sum_int
}
\ExplSyntaxOff

\sumw{10}
```

```
55
```

6.3 재귀적 정의

n까지의 합을 구하는 것은

$$S_n = n + S_{n-1} \quad (S_1 = 1,\ n \geq 2)$$

임을 이용하여 재귀적으로 정의할 수 있다.

```
def sumr(n):
    if n=1:
        return 1
    else:
        return n+sumr(n-1)
```

expl3에서도 이런 정의가 가능하다.

```
\ExplSyntaxOn
\NewDocumentCommand \sumr { m }
{
    \sum_recur:n { #1 }
}

\cs_new:Npn \sum_recur:n #1
{
    \int_compare:nTF { #1 == 1 }
    {
        1
    }
    {
        \int_eval:n { #1 + \sum_recur:n { #1 - 1 } }
```

```
    }
}
\ExplSyntaxOff

\sumr{100}
```

5050

예를 들어 expl3에서 `#1+1`이라고 하는 것은 두 수를 더하라는 의미가 아니라 이 모두가 개별 토큰임을 잊어서는 안 된다. 이들을 계산 가능한 연산식으로 취급하라는 것, 즉 int가 되도록 하라는 명령 `\int_eval:n`에 주의하여야 한다. 무슨 말이냐면, 예시한 정의에서 `\int_eval:n`을 빼먹으면 다음과 같은 결과를 얻는다.

```
\sumra{10}
```

10+10-1+10-1-1+10-1-1-1+10-1-1-1-1+10-1-1-1-1-1+10-1-1-1-1-1-1+10-1-1-1-1-1-1-1+10-1-1-1-1-1-1-1-1+1

TEX에서 꼬리재귀라 하는 것은 기본적으로 "다음 토큰을 처리하라"는 것을 말한다. 여기서 사용된 재귀호출은 이러한 재귀와는 다른 것이다. 그러므로 그 효율이 좋지만은 않고 지나치게 깊은 재귀호출이 stack overflow를 내기도 한다. 요컨대 이런 게 가능하다는 것이지 일반적 의미에서 효율적 코딩이라고 말하지 못할 경우가 많다.
또한, 이런 형식의 재귀적 함수를 정의할 때는 재귀적 함수 자신이 확장가능(expandable)하도록 정의되어야 오류를 줄일 수 있다. 앞서 xparse의 `\NewDocumentCommand`는 기본적으로 protected (또는 robust) 매크로를 만든다는 것을 지적하였다. 반면 expl3의 `\cs_new:Npn`은 명시적으로 `\cs_new_protected:N`하지 않으면 protect하지 않은 cs를 만든다.
위의 예와 같은 간단한 보기에서는 굳이 `\NewExpandableDocumentCommand`로 정의하지 않아도 되지만 명령의 확장 단계가 깊어지면 문제가 생길 수 있다. 그래서 `\cs_new:Npn`로 메인 루틴을 정의하였다.

6.4 동일검사와 case문

수(numbers), 즉 int와 fp 및 길이(dim)가 같은지를 검사해야 하는 경우가 있다. 이 세 종류의 (연산가능한) 자료형에 대해서는

`\int_compare:nTF` `\fp_compare:nTF` `\dim_compare:nTF`

로 그 동일성과 크기를 비교한다.
크기를 정할 수 없는 자료형, 즉 tl과 str에 대해서는 동일성 검사를

`\tl_if_eq:nnTF` `\tl_if_eq:NNTF`
`\str_if_eq:nnTF` `\str_if_eq:eeTF`

로 행한다. 이밖에 함수 이름이 같은지를 검사하는 `\cs_if_eq:`, skip이 동일한가를 비교하는 `\skip_if_eq:`가 있는데 이에 대해서는 여기서 더 논의하지 않는다. 한편, 위의 `...if_eq:`에 대하여는 bool 값을 돌려주는 `...if_eq_p:` 형식의 명령도 함께 존재한다.

붙임 5: tl과 cs에 대해서는 이것이 비어 있는지 정의되지 않았는지 존재하는지 등을 검사(test)하는 특별한 함수들이 있다. 추후 필요한 곳에서 언급될 것이다.

언제 tl에 대하여 비교하고 언제 str로 비교할 것인가? 일반적인 의미에서 두 "글자"가 같은가를 보려면, `\str_if_eq:nn`을 쓰는 것이 좋다. 왜냐하면 tl은 character code와 catcode가 모두 같아야 동일하다고 할 것이기 때문이다. 게다가 인자가 하나는 매크로이고 하나는 문자열일 적에 이 둘을 비교하려면 살짝 특별한 방법으로 매크로를 확장해주지 않으면 안 된다. 확장 명령을 쓰지 않고 이 둘을 직접 비교할 방법이 없다.

그러나 str는 일단 catcode를 모두 통일시켜 둔 것이다. 그러므로 charcode가 같은지만 보면 되는 것이다. 안전하게 두 문자(열)가 같은지를 검사할 수 있다. 한편 `\str_if_eq:eeTF`의 e는 중괄호 안에 들어오는 것이 매크로일 때 그 매크로를 구성하는 문자에 대하여 비교하지 말고 매크로를 해동하여 비교하도록 하는 명령이다.

가령, 주어지는 문자열에 대하여 a이거나 b이면 대문자로 식자하고 그렇지 않으면 소문자로 식자하라는 명령을 만든다고 하자.

```
\ExplSyntaxOn
\NewDocumentCommand \test { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_map_inline:Nn \l_tmpa_tl
    {
        \bool_if:nTF
            { \str_if_eq_p:nn { ##1 } { a } ||
              \str_if_eq_p:nn { ##1 } { b } }
        { \tl_upper_case:n { ##1 } }
        { \tl_lower_case:n { ##1 } }
    }
}
\ExplSyntaxOff
\test{brain}
```

BrAin

javascript 등의 switch case문에 비할 수는 없지만 종래의 LaTeX에 비하면 너무나 편리한 case 문이 expl3에서 제공된다. `\tl_case:nnTF`, `\str_case:nnTF`, `\int_case:nnTF`, `\dim_case:nnTF`가 있다. tl과 str에 대해서는 앞서 eq에 대하여 말한 바와 같다. 문자열의 동일성에 대한 문제라면 tl보다 str를 쓰는 것이 안전하다. 또한 str로 비교할 적에 매크로로 들어오는 것을 확장하여 비교하기 위한 `\str_case_e:nnTF`가 있다.
case문의 사용방식은 조금 특별하므로 주의가 필요하다. 예를 들어,

```
if x=a then processA
elseif x=b then processB
elseif x=c then processC
else processD
```

이런 가상코드를 case문으로 쓰면 대략 다음과 같이 된다. x와 a 등이 문자열이라면

```
\str_case:nnTF { x }
{
    { a } { processA }
    { b } { processB }
    { c } { processC }
}
{ }
{ process D }
```

여기서 <T> 부분은 위의 case 검사, 즉 x=a, x=b, x=c의 비교 검사가 true인 경우에 실행할 코드를 의미한다. 다르게 말하면 이 일치 검사가 성공했을 때 공통적으로 실행할 루틴을 적는다. 위의 예시에서는 공통적으로 실행할 것이 없으므로 비워두었다. 반면 <F> 부분은 위의 검사를 통과하지 못한 경우에 실행할 코드다. 일반적인 case문의 else부분에 해당하는 것. T와 F 부분은 생략가능하고 이 때는 `\str_case:nn`까지만 쓸 수 있다.
a와 b를 대문자로 바꾸는 위의 코드를 case문으로 다시 쓴 다음 예로 쉽게 이해가 될 것이다.

```
\ExplSyntaxOn
\NewDocumentCommand \test { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_map_inline:Nn \l_tmpa_tl
    {
        \str_case:nnTF { ##1 }
        {
            { a } { A }
            { b } { B }
        }
        { }
        { \tl_lower_case:n { ##1 } }
    }
}
\ExplSyntaxOff
\test{brain}
```

BrAin

위의 예에서 True일 때 실행할 코드가 없으므로 `\str_case:nnF`로 쓰고 이 부분을 없애도 된다. 또는 위의 예는 사실 대문자로 변환한다는 공통점이 있으므로

```
\str_case:nnTF { ##1 }
{
    { a } {  }
    { b } {  }
}
{ \tl_upper_case:n { ##1 } }
{ \tl_lower_case:n { ##1 } }
```

과 같이 쓰는 것도 좋다.

이번에는 다음과 같은 case문을 생각해보자.

```
case a>0 && a<10:
    <code 1>
case a>=10 && a<100:
    <code 2>
```

이것은 부등식 조건이다. `\int_compare:n`만으로는 이 조건을 지시할 수 없다. 이럴 경우에는 2019년에 추가된 함수 `\bool_case_true:n`을 활용할 수 있다.

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int { 77 }
\bool_case_true:nTF {
    { \int_compare_p:n { 0< \l_tmpa_int < 10 } } { 1 }
    { \int_compare_p:n { 10< \l_tmpa_int < 100 } } { 2 }
}
{ number~is~greater~than~0~and~less~than~100 }
{ number~is~greater~than~100 }
\ExplSyntaxOff
```

2number is greater than 0 and less than 100

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

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

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.

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.

1. 입력: `\testsum{5}`
   출력: 318
2. 입력: `\simplefact{10}`
   출력: 3628800
3. 입력: `\bin_power:n {8}`
   출력: 256
4. 입력: `\mytodec{101110}`
   출력: 46

문제

명령 \mydigit는 두 개의 인자를 받아들인다. 첫 인자는 주어지는 정수이고 두 번째 인자는 자릿수이다. #1의 #2자리 숫자를 출력하시오.

입력: \mydigit{2963}{100}
출력: 9

7 정수의 연산

\int_eval:n로 정수 표현식을 연산할 수 있다. 한편 정수 연산을 위한 함수들도 마련되어 있는데, 변수를 다룰 때는 이 편이 편하다. 예를 들어

```
\int_set:Nn \l_tmpa_int { 1 }
```

이 변수의 값을 5 증가시키려고 할 때

```
\int_set:Nn \l_tmpa_int { \l_tmpa_int + 5 }
```

이렇게 적는 것보다

```
\int_add:Nn \l_tmpa_int { 5 }
```

라고 할 수 있다.
다음은 정수 연산을 위한 함수들이다.

- \int_add:Nn 더하기
- \int_sub:Nn 빼기
- \Int_abs:N 부호 제거(절댓값)
- \int_div_truncate:nn 버림 나눗셈
- \int_div_round:nn 반올림 나눗셈
- \int_mod:nn 나머지
- \int_max:nn 큰 쪽
- \int_min:nn 작은 쪽

1부터 10까지 수를 나열하면서 3의 배수가 되면 fbox를 쳐본다.

```
\ExplSyntaxOn
\int_step_inline:nn { 10 }
{
    \int_compare:nTF { \int_mod:nn { #1 } { 3 } == 0 }
    {
        \fbox{#1},~
    }
    {
        #1,~
    }
}
\ExplSyntaxOff
```

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

위의 코드가 다른 함수나 명령의 정의부분 안에 들어간다면 #1은 ##1이 되어야 한다.

8 자릿수

2019의 십의 자리는 1이다. 주어지는 수의 특정 자리 수만을 출력하도록 해보자.

버림나눗셈을 이용하는 조작 전통적으로 이 문제의 해결책은 다음과 같다. 253의 십의 자리 숫자를 구하려면,

(a) 구하려는 십의 자리보다 한 자리 높은 100으로 준 수를 trunc하고 다시 100을 곱한다. (200)

(b) 원래 수에서 이를 빼준다. (253 – 200 = 53)

(c) 남은 수를 구하려는 자릿수 10으로 trunc한다. (5)

```
\newcount\m\newcount\n
\n=253 \m=\n \divide\m by100 \multiply\m by100
\advance\n by-\m \divide\n by10 \the\n
```

```
5
```

이 코드의 expl3 버전은 다음과 같다.

```
\ExplSyntaxOn
\NewDocumentCommand \mydigit { mm }
{
    \int_set:Nn \l_tmpa_int { \int_div_truncate:nn { #1 } { 10 * #2 } }
    \int_set:Nn \l_tmpb_int { #1 - 10 * #2 * \l_tmpa_int }
    \int_div_truncate:nn { \l_tmpb_int } { #2 }
}
\ExplSyntaxOff
\mydigit{253}{10}
```

```
5
```

이 예에서 보는 바와 같이 `\int_div_truncate:nn`이나 `\int_mod:nn`의 결과는 other 숫자의 tl로 반환된다. 이것을 수(number)처럼 다루려면 `\int_eval:n` 범위 안에 들어가야 한다. `\int_set:Nn`의 두 번째 인자 위치는 이 작용이 자동으로 이루어지므로 위의 코드는 이상없이 실행된다. 이것이 int가 아니라 tl임을 보여주는 것은 마지막 줄이다. 이 상태 그대로 숫자가 입력 스트림에 남겨진다.

tl로 보고 문자열 조작 253을 숫자가 아니라 2와 5와 3이라는 토큰의 집합으로 보고 자릿수를 추출해보자.

```
\ExplSyntaxOn
\NewDocumentCommand \mydigit { mm }
{
    \int_set:Nn \l_tmpa_int { \tl_count:n { #2 } }
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_reverse:N \l_tmpa_tl
    \tl_item:Nn \l_tmpa_tl { \l_tmpa_int }
}
\ExplSyntaxOff
\mydigit{1100}{10}
```

```
0
```

253이라는 문자열에서 10의 자리라는 것은 "뒤에서 2번째 자리"이다. 우리가 정의하는 명령의 두 번째 인자는 1, 10, 100, 이런 식으로 들어올 텐데, 이 문자열의 길이를 취하면 뒤에서 몇 번째 자리의 것을 얻어야 하는지 간단히 알게 된다. 그것이 `\l_tmpa_int`이다.
뒤에서 t번째 항목을 얻기 위해서 reverse한 다음 t번째 아이템을 취하였다. 제법 기발하게 문제를 해결한 예라고 하겠다. 그러나 이 방법으로 얻어지는 결과는 tl이라는 것을 꼭 기억하고 있어야 한다.

리스트의 특정 아이템 tl, clist, seq에 대하여 `\<type>_item:Nn` 명령이 있다. 두 번째 n인자는 언제나 정수이기 때문에 여기서 자연스럽게 `\int_eval:n`이 이루어진다. 이 함수의 실행 결과는 아이템을 입력 스트림에 남기는 것이다. 모든 아이템에 대해서는 map하지만 특정 아이템만 추려내려면 이렇게 한다. 이 함수는 원래의 리스트를 변경하지 않는다.

연습문제

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

실력 2. 두 수를 인자로 받아 그 최대공약수를 출력하는 명령 `\mygcd`를 작성하여라.

1.
입력: `\numtohangul{2019}`
출력: 이천십구

2.
입력: `\mygcd{16}{24}`
출력: 8

힌트 2: 두 수의 최대공약수를 얻는 유클리드 알고리즘의 trivial 버전은 다음과 같다. 이를 expl3로 옮겨보아라.

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

문제

인자로 주어지는 정수 이하의 모든 소수를 출력하는 명령 `\primes`를 작성하여라.

입력: `\primes{10}`
출력: 2, 3, 5, 7

다음과 같은 방법으로 소수를 구하자.

```
>>> def fnprimes(n):
        P=[]
        for i in range (2,n):
                isPrime = True
                for j in range (2, int(i ** 0.5) + 1):
                        if i % j == 0:
                                isPrime = False
                                break
                if isPrime:
                        P.append(i)
        return P

>>> fnprimes(20)
[2, 3, 5, 7, 11, 13, 17, 19]
```

9 fp 연산

TEX과 LATEX에 있는 "수"는 count와 dimension뿐이다. expl3는 fp 자료형을 도입하여 TEX에 현저히 부족한 실수 계산을 보완하려 하였다.

붙임 6: expl3가 처음 작성되기 시작하던 무렵에는 pgf의 계산 엔진을 활용하였다. 현재는 그렇지 않으나 이런 까닭에 pgfmath와 유사한 데가 남아 있다.

붙임 7: pgfmath 등장 이전에는 실수 계산을 위한 패키지들이 존재하였다. `realcalc`, `fp` 등이 이에 해당한다.

붙임 8: expl3의 fp는 유효숫자(significand) 16자리, 지수 ± 10000 범위의 수와, 부호 있는 0, 부호 있는 ∞를 포함한다.

fp 자료형의 핵심 함수는 `\fp_eval:n`이다. 인자로 주어지는 식을 parsing하여 결과를 fp형식으로 처리한다.

```
\ExplSyntaxOn
\fp_eval:n { 10/3 }
\ExplSyntaxOff
```

3.333333333333333

중요한 연산자와 부호는 다음과 같다.

- +, -, *, /, ( )
- e 부동소수점 실수 표현에 사용하는 부호이며 자연로그의 밑 e와는 관계없다. `1.666e2` = 166.6

- `inf`무한대.$+\infty$.
- `**`지수연산자.^를써도된다고하나`**`를일관되게쓰는것을권장한다.`\fp_eval:n {2**10}`=1024
- `&&`, `||`, `!` 논리 연산자 and, or, not이다. 다른 자료형에서와 달리 다음과 같은 경우도 의미를 가진다. 이 연산의 결과가 0인 것은 boolean FALSE인 것과 동일하다. 1이면 TRUE이다.

```
\ExplSyntaxOn
\fp_set:Nn \l_tmpa_fp { 2.0 }
\fp_eval:n { \l_tmpa_fp < 0.5 || \l_tmpa_fp = 2.0 }
\ExplSyntaxOff
```

1

- `?:` 3항 연산자이다. `\fp_eval:n { A ? a : B ? b : c }` 형식으로 쓸 수 있다. if A then a elseif B then b else c fi fi의 의미를 갖는다. A, B, C는 조건 연산식이고 a, b, c는 fp 값이다. (즉 일반 매크로가 a, b, c 자리에 올 수 없다.) C같은 언어에 익숙하다면 이 3항 연산자도 쉽게 이해할 수 있겠지만, 코드의 가독성이 떨어지기 때문에 남용하는 것은 권하지 않는다.

```
\ExplSyntaxOn
\fp_set:Nn \l_tmpa_fp { 2.00 }
\fp_eval:n {
    \l_tmpa_fp < 0.5 ? 1.0 :
    \l_tmpa_fp ** 2 < 3 ? 2.0 : 3.0
}
\ExplSyntaxOff
```

3

fp 함수는 `\fp_eval:n` 범위 안에서 연산할 수 있는 함수들이다.

- `exp, ln, fact` : $e^{\cdot}$, ln, factorial. 자연로그의 밑(e)을 나타내려면

```
\ExplSyntaxOn
\fp_set:Nn \l_e_fp { exp(1) }
\fp_use:N \l_e_fp
\ExplSyntaxOff
```

2.718281828459045

```
\ExplSyntaxOn
$\log \c_math_subscript_token {10} 2 = \fp_eval:n { ln(2) / ln(10) } $
\ExplSyntaxOff
```

$\log_{10} 2 = 0.3010299956639811$

- `sin, cos, tan, cot, csc, sec` : 호도법(radian) 각을 인자로 취한다.
- `sind, cosd, tand, cotd, cscd, secd` : 도(degree)를 인자로 취한다.
- `sqrt` 제곱근.
- `pi` : $\pi \simeq 3.141592653589793$.

- `round, trunc` 반올림 또는 버림.

```
\ExplSyntaxOn
\fp_eval:n { round ( 3.14159 ) } \par
\fp_eval:n { round ( 3.14159, 3 ) }
\ExplSyntaxOff
```

```
3
3.142
```

- `ceil, floor`

type 변환

① 숫자로 이루어진 tl은 확장하여 `\fp_eval:n` 하면 된다. `\fp_eval:n`의 인자 영역에 들어간 한 번 또는 두 번 확장가능한 tl은 자연스럽게 확장된다.

② 정수(int)는 `\fp_eval:n` 범위 안에서 바로 쓸 수 있다.

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int { 2 }
\fp_eval:n { sin ( ( \l_tmpa_int ** 0.4 ) pi ) }
\ExplSyntaxOff
```

```
-0.8434985587189794
```

③ `\fp_eval:n`의 결과의 소수부(decimal part)가 0이면 int에 자연스럽게 할당된다.

④ `floor, ceil` 연산의 결과는 int에 할당된다.

⑤ 일반적인 fp는 `trunc 0` 하거나 `round 0`하면 int에 할당된다.

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int { \fp_eval:n { 6/3 } }
\int_use:N \l_tmpa_int \par
\int_set:Nn \l_tmpa_int { \fp_eval:n { ceil ( 3/2 ) } }
\int_use:N \l_tmpa_int \par
\int_set:Nn \l_tmpa_int { \fp_eval:n { trunc ( 100 * sind ( 20 ) ) } }
\int_use:N \l_tmpa_int
\ExplSyntaxOff
```

```
2
2
34
```

길이(dim)와 fp의 관계는 아주 중요하므로 dim을 다루는 곳에서 자세히 연습하기로 한다.

이제 다음 문제를 풀어보자.

보조문제 1

주어진 정수의 제곱근을 넘지 않는 최대 정수를 출력하여라.

x를 넘지 않는 최대 정수를 $\lfloor x \rfloor$로 나타내기로 하면,

```
\ExplSyntaxOn
$\lfloor 2.95 \rfloor = \fp_eval:n { floor ( 2.95 ) } $
\ExplSyntaxOff
```

$\lfloor 2.95 \rfloor = 2$

이다. 또는 인자 없이 trunc하여도 같다. 이 때 truncate가 소수점 위치에서 일어나기 때문이다.

```
\ExplSyntaxOn

\cs_new:Npn \get_floor_sqrt:n #1
{
    \fp_eval:n { floor ( sqrt ( #1 ) ) }
}

\get_floor_sqrt:n { 20 }

\ExplSyntaxOff
```

4

명제

양의 정수 n, p, q에 대하여 n이 1과 자기 자신이 아닌 p와 q의 곱으로 나타낼 수 있을 때, $p < q$라면 p의 최댓값은 $\sqrt{n}$이다.

이 명제 때문에 "차례로 나누어보는" 작업을 $\lfloor \sqrt{n} \rfloor$까지만 반복하면 된다. 이를 중학생에게 증명하라고 해보았다.

Proof. 모두 양수이므로 산술-기하평균 부등식에 의하여

$$\frac{p+q}{2} \geq \sqrt{pq}.$$

① $p = q$인 경우, 좌변은

$$\frac{p+p}{2} = p$$

이고 우변은

$$\sqrt{p \times p} = p$$

이므로

$$p = \sqrt{n} \tag{1}$$

이다.
② $p < q$인 경우,

$$\frac{p+p}{2} < \frac{p+q}{2}$$

이고 산술-기하평균 부등식에 의하여

$$\frac{p+q}{2} \geq \sqrt{n}$$

으로서 $(p+q)/2$의 최솟값이 $\sqrt{n}$이다. 따라서

$$p < \sqrt{n} \leq \frac{p+q}{2}. \tag{2}$$

(1)과 (2)로부터

$$p \leq \sqrt{n}$$

이다. □

10 cs와 인자 확장

우리는 지금까지 함수를 정의하는 데 `\cs_new:Npn`을 사용해왔다. 이제 `\cs_set:Npn`을 소개하려 하는데, 이 명령은 몇 가지 용법을 가지고 있다.

(1) 이미 정의된 함수(cs)의 내용을 수정할 때. 실제로는 `\cs_new:Npn` 하지 않아도 바로 `\cs_set:Npn`할 수 있는데 그렇게 하지 않도록 유도하는 이유는 plain TEX의 `\def` 위험성과 같은 이유에서이다. 즉 그 함수가 이미 정의되어 있는지를 체크하지 않기 때문에 발생하는 위험을 줄이기 위해서 `\cs_new`를 쓰도록 한 것이다.

(2) 함수(즉 :와 인자형 지시자를 반드시 갖는 cs) 이름만이 아니라 일반 매크로의 내용을 정의하려 할 때

(3) 지역적으로(locally) 함수의 동작을 잠시 바꾸려 할 때. 그룹 안에서 `\cs_set:N`한 것은 그룹을 벗어나면 효력을 잃는다.

다음에 예를 들어본다.

```
\ExplSyntaxOn
\cs_new:Npn \foo_fn:n #1 { This~is~#1 }
\foo_fn:n { test }
\par
\cs_set:Npn \foo_fn:n #1 { You~are~#1 }
\foo_fn:n { beautiful }
\ExplSyntaxOff
```

This is test
You are beautiful

```
\ExplSyntaxOn
\cs_set:Npn \mytest #1 { Hello,~#1! }
\mytest{boys}
\ExplSyntaxOff
```

Hello, boys!

이 두 번째 사용법을 이용하면 `\NewDocumentCommand`를 쓰지 않더라도 문서 명령을 만들 수 있다. 즉

```
\ExplSyntaxOn
\cs_set_eq:NN \hello \foo_fn:n
\ExplSyntaxOff
\hello{nice}
```

This is nice

이와 같이 정의할 수 있는 것이다.
일반적인 LATEX 명령은 "풀리지 않게" 정의하는 것이 책갈피나 moving arguments에서 깨지지 않게 보호할 수 있다. 그렇기 때문에 사용자 문서 명령은 `\NewDocumentCommand`를 쓰는 것이 바람직하다. 그렇지만 아주 특별히, 풀리는 명령을 정의해야 할 필요가 있을 때, 그것도 문서 명령 비슷하게 해야 할 때 이것이

한 가지 팁이 될 수 있다. (그러나 `\cs_if_...` 명령으로 이렇게 정의된 명령을 검사하면 에러가 발생할 가능성도 있으므로 주의를 요한다.)

cs를 정의할 때 인자를 확장하려면 어떻게 하는가? 예를 들어 `\foo_fn:o`라는 형태로, 즉 들어오는 인자를 무조건 한 번 확장하여 사용하려 한다면 다음과 같이 정의한다.

```
\ExplSyntaxOn
\cs_new:Npn \foo_a:n #1
{
    \fbox{#1}
}

\cs_generate_variant:Nn \foo_a:n { V, o }
\ExplSyntaxOff
```

`\cs_generate_variant:Nn`은 모든 확장형을 다 쓸 수 있게 해주는 것은 아니다. 여기서는 다음 사항을 기억하자.

(1) 인자가 tl일 때, n을 V, o, x로 확장할 수 있고, N을 V로 확장할 수 있다.

(2) 인자가 int, fp, dim일 때, n을 V, o, x로 확장할 수 있다.

유념할 것은 예컨대 자신이 함수의 이름을 `\my_func:x`라고 지었다고 해서 바로 x확장이 일어나는 것은 아니라는 것이다. 그러므로 일단 모든 인자를 무조건 n으로 하는 함수 기본 형을 정의한 후에 이의 variant를 기술하는 것이 좋은 코딩 방법이 된다.

11 소수 구하기

원래의 문제로 돌아와서, 주어진 알고리즘을 잘 상고해보면 `break` 문이 들어 있는 것을 볼 수 있다. 그러므로 for 루프를 쓰도록 되어 있지만 우리는 map을 활용해야 한다.
n이 주어지면 그 수까지의 자연수를 담고 있는 clist를 먼저 만들기로 하자. 1은 제외한다.

```
\ExplSyntaxOn

\cs_new:Npn \build_numlist:n #1
{
    \clist_clear:N \l_tmpa_clist
    \int_step_inline:nnn { 2 } { #1 }
    {
        \clist_put_right:Nn \l_tmpa_clist { ##1 }
    }
}

\build_numlist:n { 100 }
\clist_use:Nn \l_tmpa_clist { , ~ }

\ExplSyntaxOff
```

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100

그리고, 주어진 수의 $\lfloor x \rfloor$까지의 clist도 하나 만든다.

```
\ExplSyntaxOn

\cs_set:Npn \build_numlist:n #1
{
    \clist_clear:N \l_tmpa_clist
    \int_step_inline:nnn { 2 } { #1 }
    {
        \clist_put_right:Nn \l_tmpa_clist { ##1 }
    }

    \clist_clear:N \l_tmpb_clist
    \int_step_inline:nnn { 2 } { \get_floor_sqrt:n { #1 } }
    {
        \clist_put_right:Nn \l_tmpb_clist { ##1 }
    }
}

\build_numlist:n { 100 }
a~:~\clist_use:Nn \l_tmpa_clist { , ~ } \par
b~:~\clist_use:Nn \l_tmpb_clist { , ~ }

\ExplSyntaxOff
```

a : 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100
b : 2, 3, 4, 5, 6, 7, 8, 9, 10

처음에 제시한 알고리즘을 구현하면 다음과 같다.

```
\ExplSyntaxOn
\cs_set:Npn \get_floor_sqrt:n #1
{
    \fp_eval:n { floor ( sqrt ( #1 ) ) }
}

\cs_set:Npn \build_numlist:n #1
{
    \clist_clear:N \l_tmpa_clist
    \int_step_inline:nnn { 2 } { #1 }
    {
        \clist_put_right:Nn \l_tmpa_clist { ##1 }
    }

    \clist_clear:N \l_tmpb_clist
    \int_step_inline:nnn { 2 } { \get_floor_sqrt:n { #1 } }
    {
        \clist_put_right:Nn \l_tmpb_clist { ##1 }
    }
}

\cs_new:Npn \fn_primes:n #1
{
    \clist_gclear:N \g_tmpa_clist

    \build_numlist:n { #1 }

    \clist_map_inline:Nn \l_tmpa_clist
```

```
    {
        \bool_gset_true:N \g_tmpa_bool
        \int_gset:Nn \g_tmpa_int { ##1 }
        \clist_map_function:NN \l_tmpb_clist \sub_fn:n
        \bool_if:NT \g_tmpa_bool
        {
            \clist_put_right:Nn \g_tmpa_clist { ##1 }
        }
    }

    \clist_use:Nn \g_tmpa_clist { , ~ }
}

\cs_new:Npn \sub_fn:n #1
{
    \bool_if:nT
    {
        \int_compare_p:n { \int_mod:nn { \g_tmpa_int } { #1 } == 0  }
        &&
        \int_compare_p:n { \g_tmpa_int != #1 }
    }
    {
        \bool_gset_false:N \g_tmpa_bool
        \clist_map_break:
    }
}

\cs_set:Npn \fnprimes #1
{
    \fn_primes:n { #1 }
    {}~( \clist_count:N \g_tmpa_clist )
}

\ExplSyntaxOff

\fnprimes{21}
```

2, 3, 5, 7, 11, 13, 17, 19 (8)

어찌 됐든 답을 구할 수는 있었지만 이 방법은 다음 두 가지 때문에 좋은 해결책이 못 된다. (1) clist에 넣을 수 있는 아이템의 개수에 제한이 있다. (2) clist 또는 seq에 정수를 입력하는 것이 효율적이지 못하다. 구하려는 수가 커지면 clist 입출력에 너무 많은 시간이 소요된다.
만약 `\int_step_...`을 중간에 중단할 수 있는 break 명령이 있다면 어떻게 할 수 있을까?

```
\ExplSyntaxOn

\cs_new:Npn \step_fn_prime:n #1
{
    \bool_gset_true:N \g_tmpa_bool
    \int_step_inline:nnn {2} { \fp_eval:n { floor ( sqrt ( #1 ) ) } }
    {
        \int_compare:nT { \int_mod:nn { #1 } { ##1 } == 0 }
        {
            \bool_gset_false:N \g_tmpa_bool
            \esg_int_step_break:
        }
    }
```

```
    \bool_if:NT \g_tmpa_bool
    {
        \clist_gput_right:Nn \g_tmpa_clist { #1 }
    }
}

\cs_new:Npn \fn_new_primes:n #1
{
    \clist_gclear:N \g_tmpa_clist

    \int_step_function:nnN {2} { #1 } \step_fn_prime:n

    \clist_use:Nn \g_tmpa_clist { ,~ }
}

\fn_new_primes:n { 25 }

\ExplSyntaxOff
```

2, 3, 5, 7, 11, 13, 17, 19, 23

이것은 처음에 제시한 python 코드를 거의 그대로 옮긴 것이다. `\esg_int_step_break:` 명령은 따로 준비하였다.

연습문제

기본 1. 본문의 예제는 루프를 탈출하기 위하여 `\clist_map`을 활용하였다. 그런데 while do를 쓰면 clist mapping을 이용하지 않아도 루프의 탈출 조건을 만들 수 있다. 이를 이용하여 같은 알고리즘을 구현할 수 있겠는가?

발전 2. 주어진 수가 소수인지를 검사하는 다음과 같은 알고리즘이 있다. 이를 expl3로 구현할 수 있겠는가?

```
>>> def isPrime(n):
        if (n<=1):
                return False
        if (n<=3):
                return True
        if (n%2 == 0 or n%3 == 0):
                return False
        i=5
        while (i*i <= n ):
                if (n%i == 0 or n%(i+2) == 0):
                        return False
                i = i+6
        return True

>>> if (isPrime(43)):
        print('prime')
else:
        print('not a prime')


prime
```

실력 3. KTUG 게시판 :235888 글에 소인수분해 알고리즘이 소개되어 있다. 이를 바탕으로 다음 순서로 문제를 해결하여라.

① 두 수를 인자로 받아서 최대공약수를 구하여라.

② 최대공약수를 소인수분해하여 결과를 clist나 seq에 저장하여라.

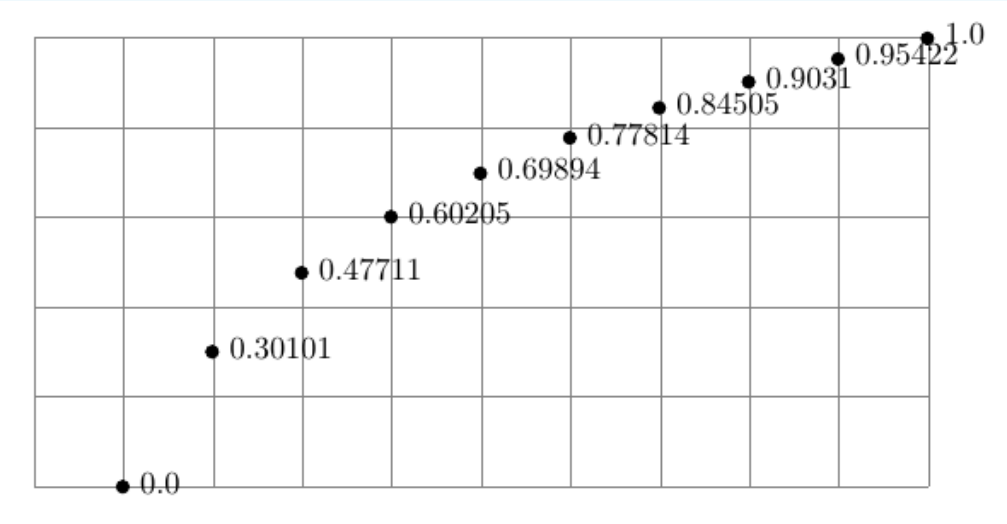
③ 최대공약수의 소인수를 취하여 차례로 두 수를 나누어가면서 몫(quotient)의 변화 과정을 clist나 seq에 저장하여라.

④ 준비된 세 개의 clist (seq)를 이용하여 다음 그림과 같이 출력하여라.

2	16	24
2	8	12
2	4	6
	2	3

문제

1부터 10까지 정수의 상용로그 값을 계산하여 좌표평면에 다음 그림과 같이 나타내어라.



먼저, 1부터 10까지 정수의 상용로그 값을 도표로 작성해보자.

```
\ExplSyntaxOn
\int_step_inline:nn { 10 }
{
    #1:~\fp_eval:n { ln ( #1 ) / ln ( 10 ) } \par
}
\ExplSyntaxOff
```

1: 0
2: 0.3010299956639811
3: 0.4771212547196625
4: 0.6020599913279625
5: 0.6989700043360185
6: 0.7781512503836435
7: 0.8450980400142566
8: 0.9030899869919435
9: 0.9542425094393246
10: 1

소수 네째자리까지만 표현하고 싶어서 round 함수를 써보면 다음처럼 된다. `round (0.30102999, 4)`와 같이 round할 위치를 `,4`로 이어 지시해야 하는 것에 주의한다.

```
\ExplSyntaxOn
\int_step_inline:nn { 10 }
{
    #1:~\fp_eval:n { round ( ln ( #1 ) / ln ( 10 ) , 4) } \par
}
\ExplSyntaxOff
```

1: 0
2: 0.301
3: 0.4771
4: 0.6021
5: 0.699

```
6: 0.7782
7: 0.8451
8: 0.9031
9: 0.9542
10: 1
```

반올림은 원하는 자리에서 잘 일어났는데 수의 표현이 가지런하지 않다. 그 이유는 fp 표현식이 끝자리 0를 모두 제거하기 때문이다.
그래서 `\format_num:n` 함수를 하나 정의하였다. 이것은 들어오는 소수의 길이를 6자리로 보고 필요한 0를 채워넣은 것이다.

```
\ExplSyntaxOn

\cs_new:Npn \format_num:n #1
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \str_if_in:NnF \l_tmpa_tl { . }
    {
        \tl_put_right:Nn \l_tmpa_tl { .0000 }
    }

    \int_compare:nT { \tl_count:N \l_tmpa_tl < 6 }
    {
        \int_step_inline:nn { 6 - \tl_count:N \l_tmpa_tl }
        {
            \tl_put_right:Nn \l_tmpa_tl { 0 }
        }
    }

    \l_tmpa_tl
}

\cs_generate_variant:Nn \format_num:n { V, x }

\format_num:n { 2.1 }
\ExplSyntaxOff
```

```
2.1000
```

현재 정의한 `\format_num:n`은 일반적인 용도에 쓸 수 없다. 왜냐하면 정수 부분이 단 한 자리인 소수만을 표현할 수 있고 소수 부분은 무조건 4자리로 맞추고 있기 때문이다.
예컨대 `\formatnum[3]{123.1}`과 같이 입력하면 `123.100`으로 표현해주는 함수를 정의하는 것은 스스로 해보는 즐거움을 위하여 미루어두겠다.
1에서 10까지의 상용로그를 소수 네째자리까지 표현하는 데는 이것으로 가능하니까 여기서는 그냥 쓰도록 하였다.

```
\ExplSyntaxOn
\cs_new:Npn \print_log_value:n #1
{
    \ensuremath{#1} \c_alignment_token
    \ensuremath{
        \format_num:x { \fp_eval:n { round ( ln (#1) / ln (10), 4 ) } }
    }
    \tabularnewline \hline
}
```

```
\begin{tabular}{|r|c|}
\hline
$x$ & $y$ \\ \hline
\int_step_inline:nn { 9 }
{
    \print_log_value:n { #1 }
}
\ensuremath{10} \c_alignment_token
\ensuremath{
    \format_num:n { 1 } }
\\ \hline
\end{tabular}
\ExplSyntaxOff
```

x	y
1	0.0000
2	0.3010
3	0.4771
4	0.6021
5	0.6990
6	0.7782
7	0.8451
8	0.9031
9	0.9542
10	1.0000

12 Ti*k*Z와 expl3

현 시점에서 "그리기"의 사실상 표준인 Ti*k*Z를 expl3 syntax 범위 안에서 쓰기 위해서 다음 두 가지를 주의하여야 한다.

(1) Ti*k*Z 옵션 등에 나타나는 space를 반드시 명시적으로 (틸데로써) 입력하여야 한다.
`\tikz[rounded~corners=3pt]`

(2) 콜론 문자를 직접 입력하여서는 안 된다. 이럴 때를 위하여 `\c_colon_str`이라는 string 상수가 정의되어 있다.

(3) Ti*k*Z 환경과 expl3는 서로 다른 확장 규칙을 가지고 있다. expl3로 작성된 함수가 Ti*k*Z 환경 안에서 성공적으로 풀리지 않을 수 있다. 이럴 때에는 Ti*k*Z 환경을 ExplSyntax 범위 밖에 두는 것을 고려해보아야 한다.

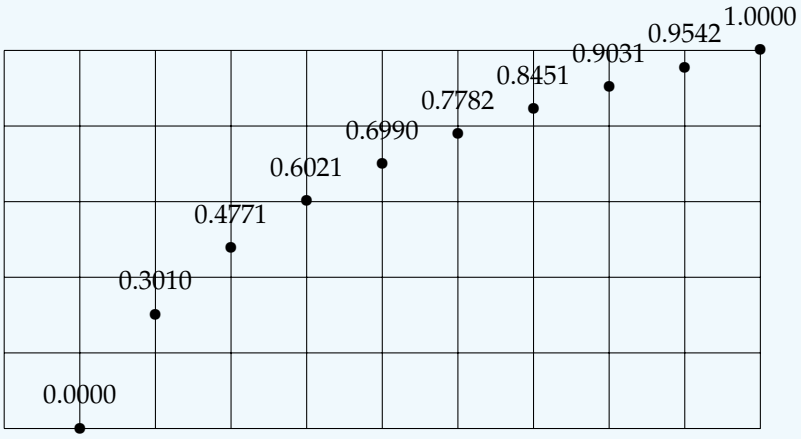
(4) Ti*k*Z 환경 내부에서 반복문을 실행해야 한다면 `\foreach`를 쓰는 것이 대체로 안전하다.

(5) 될 수 있으면 expl3로 행하는 계산이나 함수의 확장은 Ti*k*Z 환경 외부에서 실행하고 결과를 expandable한 tl이나 매크로로 Ti*k*Z 함수에 넘겨주도록 코드를 작성하는 것이 좋다.

이 몇 가지를 주의하면 ExplSyntax 안에서 Ti*k*Z를 사용하는 것이 가능하다.

```
\ExplSyntaxOn
\cs_new:Npn \mylog:n #1
{
    \fp_eval:n { round ( ln ( #1 ) / ln ( 10 ) , 4 ) }
}
\cs_set_eq:NN \mylog \mylog:n
```

```
\begin{tikzpicture}
\draw (0,0) grid (10,5);
\foreach \x in {1,2,...,10}
    \node at (\x, 5*\mylog{\x}) [label={\format_num:x { \mylog{\x}} }]
     ↪  {$\bullet$};
\end{tikzpicture}
\ExplSyntaxOff
```



확장 (4) `\exp_args:` 함수는 그 이름에서 알 수 있는 바와 같이 arguments를 expand하는 데 쓰는 것이다. 그런데 arguments가 아니라 멀쩡한 매크로가 있는 자리에서 그 매크로 자체를 (`\expandafter` 처럼) 확장하고자 한다면 어떻게 해야 하는가?
이럴 때 쓰는 것이 `\exp_last_unbraced:` 함수이다. 예를 들어보자.

```
\ExplSyntaxOn
\begin{tikzpicture}
\draw (0,0) -- (3,3);
\end{tikzpicture}
\ExplSyntaxOff
```

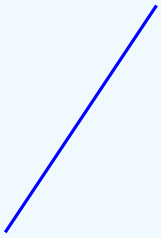
이 경우에, 선분의 끝점을 매크로로 전달하는 상황을 가정한다.

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { [very~thick,blue] }
\cs_set:Npn \my_point:n #1 { (#1,\int_eval:n { #1+1 } ) }
\tl_set:Nx \l_tmpb_tl { (0,0) -- \my_point:n { 2 } }
\begin{tikzpicture}
\draw \l_tmpa_tl \l_tmpb_tl ;
\end{tikzpicture}
\ExplSyntaxOff
```

(이 샘플은 별도로 확장 안 해도 잘 동작하지만 설명을 위해 예를 드는 것이므로) 여기서 `\l_tmpa_tl`와 `\l_tmpb_tl` 매크로를 확장하여야 할 적에 다음과 같이 한다.

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { [very~thick,blue] }
\cs_set:Npn \my_point:n #1 { (#1,\int_eval:n { #1+1 } ) }
\tl_set:Nx \l_tmpb_tl { (0,0) -- \my_point:n { 2 } }
```

```
\begin{tikzpicture}
\exp_last_unbraced:NNx \draw \l_tmpa_tl \l_tmpb_tl ;
\end{tikzpicture}
\ExplSyntaxOff
```



요약하면, 브레이스({})로 둘러싸여 전달되는 부분을 확장하려면 `\exp_args:`를 쓰고 단일 매크로를 확장하려 할 때는 `\exp_last_unbraced:`를 쓴다고 해도 좋다.
이미 강조한 바이지만, 예를 들어

```
\my_func:n { \l_tmpa_tl }
```

이런 상황에서 `\l_tmpa_tl`이 단 한 개의 문자로 이루어져 있을 때 `\my_func:n \l_tmpa_tl`라고 쓰고 싶은 유혹을 충분히 느끼겠지만 확장 관련 문제가 복잡하게 얽힐 때에 대비하여 n 아규멘트라면 중괄호를 꼭 써주어야 한다.

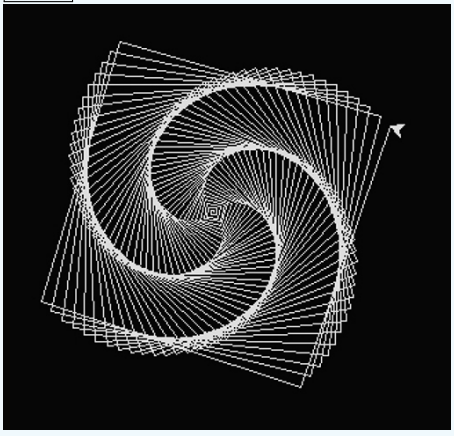
연습문제

기본 1. 가로 10cm, 세로 10cm이고 상하좌우 여백이 1.5cm, 모두 30페이지를 가진 pdf 문서를 작성한다. 매 페이지마다 현재 페이지가 전체 페이지수에 대하여 몇 %인지를 표시하고 페이지 번호가 5의 배수가 되는 때 페이지의 배면색상(background color)를 cyan으로 하여라.

기본 2. 위의 pdf 문서 각 페이지의 중앙에 반지름 3cm인 원을 그리고 진행비율(현재페이지/전체페이지)을 붉은 색으로 표시하는 progress pie를 그려라.

기본 3. 중학교 수학 교과서의 부록으로 "삼각비표"가 있다. 이 표의 일부 (0° 부터 25° 까지)를 되도록 예쁘게 작성하여라.

실력 4. 다음 그림을 그려보아라. 배경색은 별도로 지정하지 않아도 좋다.



문제

현재 조판하고 있는 페이지의 메트릭을 mm 단위로 보여주는 명령 `\currentmetric`을 작성하여라.

입력: `\currentmetric{textwidth}`
출력: 184.4mm

13 TEX의 dimension과 expl3의 dim 데이터타입

TEX의 길이 단위 연산자 TEX은 내부적으로 모든 길이를 sp로 계산한다. 사용자 인터페이스에서는 pt를 기본 단위로 쓴다. 1pt = 65536sp이다. 다음은 미리 정의된 길이 단위 연산자들이다.

sp, pt, mm, cm, in
bp, pc, dd, cc, nd, nc

폰트에 따라 가변적 값을 가지는 두 가지 단위가 있다.

em, ex

TEX에서 사용하는 point는 소위 "작은 포인트"로서 1/72.27인치에 해당한다. 반면 PostScript 포인트를 "큰 포인트"라고 하고 이것은 1/72인치이다. 이에 해당하는 단위는 bp이다.
em과 ex는 폰트의 디자인에 관련되는 단위이다. em은 "대문자 M의 width"라는 의미에서 나온 말이지만 현대의 많은 폰트에서 대문자 M은 실제로 1em을 넘거나 모자라는 경우가 많다. 이것은 전적으로 디자인 문제라고 생각하면 된다. 글자를 디자인하는 정사각형 디자인 박스의 가로세로 폭을 1em이라고 하는데, 이를 1000으로 설정한 다음 이 정사각형 안에 글자 모양을 그려넣는 것이 일반적이다. 현재 이 문서의 영문자는 TeX Gyre Pagella를 쓰고 있는데 M자의 폭과 1em의 길이를 구해보면

```
\ExplSyntaxOn
\hbox_set:Nn \l_tmpa_box { M }
\dim_eval:n { \box_wd:N \l_tmpa_box } \\
\dim_eval:n { 1em }
\ExplSyntaxOff
```

9.46pt
10.0pt

즉, 폰트 사이즈가 10pt일 때 1em은 10pt이지만 M자의 width는 디자인에 따라 달라진다.
한편 ex는 x자의 높이라는 의미인데, 이 역시 디자인 상의 단위이다. 라틴 문자는 4개의 선을 그어놓고 알파벳을 디자인한다고 생각할 때, 아래에서 두 번째 줄이 베이스라인이고 베이스라엔에서 그 다음 세 번째 줄까지의 길이를 ex라고 하는 것이다. em과 달리 이 길이는 폰트마다 다 다르고 폰트 사이즈에 따라서도 달라진다. 실제 식자되는 x자의 높이와 거의 같다.

```
\ExplSyntaxOn
\hbox_set:Nn \l_tmpa_box { x }
\dim_eval:n { \box_ht:N \l_tmpa_box } \\
\dim_eval:n { 1ex }
\ExplSyntaxOff
```

```
4.69pt
4.69pt
```

dim 데이터타입 `\dim_eval:n`의 인자로 dim 표현식이 오는 것은 다른 "수"의 경우와 똑같다. 내부적으로 dim은 fp와 비슷한 모양을 하고 있지만 그것은 우리가 interface 단위로서 pt를 채택하고 있어서 그럴 뿐이고 실제로는 sp 단위의 정수 연산을 하고 있다.
dim 변수는 길이를 요구하는 모든 곳에 적용가능하다.
dimension 표현식은 반드시 단위 연산자가 붙는다. 예를 들면

```
\dim_set:Nn \l_tmpa_dim { 29.92 }
```

이것은 단위가 붙지 않았다는 오류를 낸다. 그렇다면 fp 표현식에 단위 연산자를 붙이면 dim에 할당이 될까?

```
\ExplSyntaxOn
\dim_set:Nn \l_tmpa_dim { \fp_eval:n { 19.6 / 2.7 } pt }
\rule{\l_tmpa_dim}{5pt}
\ExplSyntaxOff
```


일반적으로 말해서 fp보다는 dim 계산이 빠르다. 그러므로 위와 같은 것은 가능은 한데 dim으로 계산할 수 있다면 그것이 더 좋다.
dim 표현식은 fp만큼 유연하지 않다. 예를 들면

```
\ExplSyntaxOn
\dim_set:Nn \l_tmpa_dim { 10pt }
\dim_eval:n { 2\l_tmpa_dim + 4pt } \\
\dim_eval:n { \l_tmpa_dim * 2 + 4pt }
\ExplSyntaxOff
```

```
24.0pt
24.0pt
```

이와 같이 "계수에 의한 곱연산"이 가능하지만

```
\ExplSyntaxOn
\dim_eval:n { 0.333*2*\l_tmpa_dim + 2pt }
\ExplSyntaxOff
```

곱셈을 연산처럼 중첩해서 쓰지 못한다.
나눗셈은 다음과 같이 쓴다. `\dim_ratio:nn` 함수는 인자 두 개를 다 dim으로만 받아들여서 fp 결과를 되돌리기 때문에 단위에 주의하여야 한다.

```
\ExplSyntaxOn
\dim_eval:n { 10pt / 2 } \\
\dim_eval:n { 10pt * \dim_ratio:nn {10pt} {2pt} }
\ExplSyntaxOff
```

5.0pt
50.0pt

dim 표현식은 확장 순서에 매우 민감하다. 예를 들면

```
\dim_eval:n { 10pt * \dim_ratio:nn {10pt} {2pt} }
```

이것은 오류를 일으키지 않지만

```
\dim_eval:n { \dim_ratio:nn {10pt} {2pt} * 10pt }
```

이것은 evaluate에 실패하였다는 메시지를 볼 수 있다. 그 이유는

```
\dim_eval:n { 2 * 20pt }
```

이것은 실패이지만

```
\ExplSyntaxOn
\dim_eval:n { 20pt * 2 }
\ExplSyntaxOff
```

40.0pt

이것은 성공하는 것과 정확하게 같다.

길이를 fp로 계산하기 그래서 가끔 아주 복잡한 길이 계산을 해야 할 때는 단위를 다 떼어놓고 fp로 계산한 다음 그 결과에 단위를 다시 붙여주는 것이 (코딩하기) 편할 때가 있다.
아주 오랜 옛날부터 `\strip@pt`라는 트릭이 있었다.

```
\makeatletter
\strip@pt\textwidth
\makeatother
```

420.70206

이렇게 해놓고 이것을 fp처럼 계산하겠다는 것이지. 이를 이용하여 다음 문제를 풀어보자.

보충문제

현재 폰트 현재 폰트사이즈에서 알파벳 대문자 M의 폭/높이(width/height)의 값을 나타내어라.

expl3를 모를 때 어떻게 했는지 기억을 되살려보면,

```
\newsavebox\M \sbox\M{M}
\newdimen\Mwidth\newdimen\Mheight
\Mwidth=\wd\M \Mheight=\ht\M
\the\Mwidth, \the\Mheight \\
\makeatletter
\strip@pt\Mwidth, \strip@pt\Mheight
\makeatother
```

9.46pt, 6.92pt
9.46, 6.92

이제 실수 나눗셈을 하면 되는데 이것은 fp나 realcalc 또는 calc 패키지를 이용하여 할 수 있을 듯하다.

expl3로는 아마 이렇게 할 수 있지 않을까?

```
\ExplSyntaxOn
\hbox_set:Nn \l_tmpa_box { M }
\dim_set:Nn \l_tmpa_dim { \box_wd:N \l_tmpa_box }
\dim_set:Nn \l_tmpb_dim { \box_ht:N \l_tmpa_box }
\fp_eval:n { \dim_ratio:nn { \l_tmpa_dim } { \l_tmpb_dim } }
\ExplSyntaxOff
```

1.367053355060208

참고, hbox와 dimension 아까부터 \hbox_. . . 어쩌고 하는 명령이 나오는데 box 데이터타입에 대하여 따로 상세히 배우게 된다.
지금은 다음과 같은 것을 능숙하게 할 수 있도록 해두자.
다른 타입과 마찬가지로 네 개의 scratch 변수가 주어져 있다. 지금은 vbox에 대하여 신경쓸 것 없이 hbox만을 알아두면 된다. hbox란 horizontal mode의 box라는 뜻이다.
여기에 식자할 material을 집어 넣는 것은

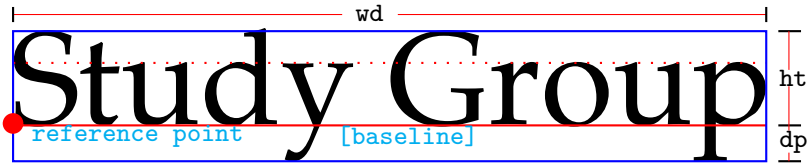
```
\hbox_set:Nn \l_tmpa_box { material }
```

이것을 typeset 하려면

```
\box_use:N \l_tmpa_box
```

주어진 box의 폭(width), 높이(height), 깊이(depth)는 다음과 같이 측정하는데 그 결과는 dim이다.

```
\box_wd:N \l_tmpa_box
\box_ht:N \l_tmpa_box
\box_dp:N \l_tmpa_box
```



totalheight는 ht+dp를 말한다.

```
\ExplSyntaxOn
\hbox_set:Nn \l_tmpa_box { Study~Group }
width:~\dim_eval:n { \box_wd:N \l_tmpa_box } \\
height:~\dim_eval:n { \box_ht:N \l_tmpa_box } \\
depth:~\dim_eval:n { \box_dp:N \l_tmpa_box } \\
totalheight:~
\dim_set:Nn \l_tmpa_dim { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
\dim_use:N \l_tmpa_dim
\ExplSyntaxOff
```

width: 57.59pt
height: 7.26pt
depth: 2.82999pt

```
totalheight: 10.08998pt
```

지금 이 긴 이야기의 결론은 사용자 레벨에서 "단위가 제거된 dim은 fp"라는 것이다. 그렇다면 dim의 단위를 제거하려면 어떻게 하는가?
`\dim_to_decimal:n`은 모든 길이를 pt로 환산한 다음 단위를 제거해준다. 이것은 `\strip@pt`의 expl3 판이라고 해도 좋겠다. 단 이 명령은 소수 다섯째 자리까지 유효한 근삿값이다.

```
\ExplSyntaxOn
\dim_to_decimal:n { \textwidth }
\ExplSyntaxOff
```

```
420.70206
```

현재 페이지의 종횡비(세로/가로)를 구해보자.

```
\ExplSyntaxOn
\fp_eval:n
  {
     \dim_to_decimal:n { \paperheight }
     /
     \dim_to_decimal:n { \paperwidth }
  }
\ExplSyntaxOff
```

```
1.4142857063958
```

A4 페이퍼는 접어도 종횡비가 유지되는 판형이다. 즉 세로 길이가 가로 길이의 $\sqrt{2}$배이다.

단위 간의 변환 `\dim_to_decimal_in_unit:nn` 함수가 제일 요긴하다. 두 번째 인자에 표시하고 싶은 단위를 써주면 된다. 10포인트가 몇 mm인지를 나타내려면

```
\ExplSyntaxOn
\dim_to_decimal_in_unit:nn { 10pt } { 1mm }
\ExplSyntaxOff
```

```
3.51462
```

이 함수 연산의 결과는 dim 형이 아니라는 점을 기억해야 한다. 단위가 따로 붙어 있지 않다. 그러므로 fp에 할당하여 연산할 수 있으며 tl에 할당하는 것도 가능하다. 표현 시에는 mm를 적어주어야 한다.

dim 조건식 `\int_compare:nTF`와 마찬가지로 `\dim_compare:nTF`가 있다.

```
\ExplSyntaxOn
\dim_set_eq:NN \l_tmpa_dim \textwidth
\dim_set_eq:NN \l_tmpb_dim \linewidth

\dim_compare:nTF { \l_tmpa_dim - \l_tmpb_dim == \c_zero_dim }
{
    textwidth == linewidth
}
{
    textwidth != linewidth
```

```
}
\ExplSyntaxOff
```

textwidth==linewidth

길이를 비교하는 것은 문제가 없지만 위의 경우처럼 0보다 큰가 작은가를 비교할 때 자주 하는 실수가

```
\dim_compare:nTF { \l_tmpa_dim == 0 }
```

과 같이 그냥 0와 비교하려 하는 것이다. 이것은 실패. 반드시 0pt와 비교하여야 한다. 이것이 실수하기 쉬우므로

```
\c_zero_dim
```

이라는 상수를 쓰는 것이 좋다. 이것은 그냥 0pt이다.
이밖에 while do와 같은 반복문도 있으므로 int가 아니라 길이가 문제일 때는 dim 자료형을 쓴다고 생각하여도 무방하다.

이제 주어진 문제를 해결해보자. 인자로 들어오는 길이 변수 이름이 만약 존재하지 않는 것이면 이를 에러처리하는 것을 포함한다.
소수 다섯째 자리가 너무 길고 눈에 들어오지 않으니까 최대 소수 둘째 자리까지만 보여주기로 하자.

```
\ExplSyntaxOn
\NewDocumentCommand \currentmetric { m }
{
    \dim_if_exist:cTF { #1 }
    {
        \calc_metric_fn:n { #1 }
    }
    {
        \texttt{\bs #1}~is~invalid
    }
}

\cs_new:Npn \calc_metric_fn:n #1
{
    \dim_set_eq:Nc \l_tmpa_dim { #1 }

    \texttt{\bs #1} $\,=\,$

    \fp_eval:n { round ( \dim_to_decimal_in_unit:nn { \l_tmpa_dim } { 1mm } ,
     ↪  2 ) }
    \,mm
}

\ExplSyntaxOff

\currentmetric{hello}

\ExplSyntaxOn
\dim_new:N \hello \dim_set:Nn \hello { .33\textwidth }
\ExplSyntaxOff

\currentmetric{hello}

\currentmetric{foremargin}
```

```
\hello is invalid
\hello = 48.79 mm
\foremargin = 25.4 mm
```

간단하지만 의외로 요긴한 명령이다. printlen이라는 패키지가 길이 단위를 제어할 수 있게 해주는데 생각해보면 패키지까지 써야 하는지 의문이기는 하다.

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

기본 2. LaTeX에 `\settowidth`와 `\settoheight`, `\settodepth`라는 명령이 있다. 이의 사용법은 다음과 같다.

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

39.75pt

이것과 동일한 작용을 하는 `\SettoWidth`, `\SettoHeight`, `\SettoDepth`를 expl3로 만들고 `\SettoTotalheight`도 작성해보아라.

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

> 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.

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.

font size command	value of em
\scriptsize	7.0pt
...	...
\large	10.95pt
...	...

문제

인자로 주어지는 각 글자를 `\colorbox`에 넣어라. 배경색상을 임의로 설정하고 전경색상은 그 반전색으로 하라. 띄어쓰기는 무시한다. 칼라 모델은 xcolor의 rgb나 HTML을 사용하여라.

입력: `\mytest{우리나라}`
출력: 우리나라

14 TEX의 가상난수

원래 TEX에는 난수생성기가 없었다. TEX 엔진 자체에서 난수를 생성할 수 있게 된 것은 pdfTEX이 처음이었다. 그 이전에는 random.tex이라는 작은 유틸리티, lcg 패키지 등이 난수를 흉내내었고, pgf 계산 엔진이 개발된 후에는 pgfmath에서 난수를 생성해주게 되었다.
expl3의 난수 엔진은 pgfmath의 것을 쓰지 아니하고 엔진 자체의 primitive를 활용한다. pdfTEX에는

```
\pdfnormaldeviate
\pdfuniformdeviate
```

라는 두 개의 primitive가 있으며 LuaTEX과 XETEX에는

```
\normaldeviate
\uniformdeviate
```

가 마련되어 있다. 특히, XETEX의 난수 프리미티브는 2019년 버전에서 처음으로 도입된 것이다. TEX Live 2018까지는 이 기능을 XETEX에서 사용할 수 없다.
정규분포 난수 생성 명령인 normaldeviate에 대해서는 나중에 기회가 닿으면 살펴보기로 하고 우리는 uniform deviate에 대해서만 다루겠다.
따로 seed를 정해주지 않으면 (다른 언어들이 그리 하듯이) 시스템 시간을 seed로 하여 가상 난수를 생성한다. (seed를 설정하거나 재설정하는 것이 가능하지만 이 문제는 당분간 논외로 한다.)
프리미티브를 직접 사용하는 예를 들어보겠다.

```
\ifPDFTeX
    \let\uniformdeviate\pdfuniformdeviate
\fi
\uniformdeviate 100
```

90

`\uniformdeviate n`이 생성하는 난수는 $0 \le x < n$ 범위의 정수이다. 이 프리미티브들은 *The Art of Computer Programming*의 제2권 3장 6절에 나오는 내용을 METAPOST로 구현한 것이다. expl3는 전적으로 각 엔진의 프리미티브에 의존한다.
정수형 난수를 얻는 함수는 다음과 같다.

```
\ExplSyntaxOn
\int_rand:n { 100 } \\
\int_rand:nn { 10 } { 20 }
\ExplSyntaxOff
```

61
20

앞서 본 프리미티브 자체와는 달리 expl3 함수는 주어진 수를 포함한다. 인자가 두 개인 함수가 만드는 난수의 범위는 $n_1 \le x \le n_2$이다. `\int_rand:n {a}`는 `\int_rand:nn {1}{a}`와 같다.
0이상 1미만의 실수형 난수를 얻는 방법은 다음과 같다.

```
\ExplSyntaxOn
\fp_eval:n { rand () }
\ExplSyntaxOff
```

0.2720112006593367

해상도는 fp 자체의 significand와 동일하게 10^{-16}이다. fp 자료형에도 정수형 난수를 얻는 randint가 있지만 정수 계산은 `\int_rand:`가 더 빠르고 좋다. 주의할 것은 fp의 rand() 함수는 위쪽 경계 1을 포함하지 않는다는 점이다.

주어진 문제를 해결하기 위해 세 개의 int 변수를 만들고 이 각각에 random으로 100 이하의 정수값을 주기로 한다. 이 값을 100으로 나누어서 R, G, B의 각 색상값으로 삼아보겠다. fp random을 쓰지 않고 int random을 쓰는 이유는 무엇보다 그것이 빠르기 때문이고 색상 표현을 위해 R, G, B 각각 100분할 정도면 충분하다고 생각하기 때문이다.

```
\ExplSyntaxOn
\int_new:N \l_r_int
\int_new:N \l_g_int
\int_new:N \l_b_int
\tl_new:N \l_r_tl
\tl_new:N \l_g_tl
\tl_new:N \l_b_tl

\cs_new:Nn \gen_rand_color:
{
    \int_set:Nn \l_r_int { \int_rand:nn { 0 } { 100 } }
    \int_set:Nn \l_g_int { \int_rand:nn { 0 } { 100 } }
    \int_set:Nn \l_b_int { \int_rand:nn { 0 } { 100 } }
    \tl_set:Nf \l_r_tl { \fp_eval:n { \l_r_int / 100 } }
    \tl_set:Nf \l_g_tl { \fp_eval:n { \l_g_int / 100 } }
    \tl_set:Nf \l_b_tl { \fp_eval:n { \l_b_int / 100 } }

    \exp_args:Nnnx \definecolor{back}{rgb}{\l_r_tl,\l_g_tl,\l_b_tl}

    \int_set:Nn \l_r_int { 100 - \l_r_int }
    \int_set:Nn \l_g_int { 100 - \l_g_int }
    \int_set:Nn \l_b_int { 100 - \l_b_int }
    \tl_set:Nf \l_r_tl { \fp_eval:n { \l_r_int / 100 } }
    \tl_set:Nf \l_g_tl { \fp_eval:n { \l_g_int / 100 } }
    \tl_set:Nf \l_b_tl { \fp_eval:n { \l_b_int / 100 } }

    \exp_args:Nnnx \definecolor{fore}{rgb}{\l_r_tl,\l_g_tl,\l_b_tl}
}

\gen_rand_color:
back:~{\color{back}\rule{20pt}{10pt}}\\
fore:~{\color{fore}\rule{20pt}{10pt}}

\ExplSyntaxOff
```

back:
fore:

반복해서 같은 유형의 문장을 계속 쓰는 것이 싫은 사람도 있을 것이다.

```
\tl_set:Nf \l_r_tl { \fp_eval:n { \l_r_int / 100 } }
\tl_set:Nf \l_g_tl { \fp_eval:n { \l_g_int / 100 } }
\tl_set:Nf \l_b_tl { \fp_eval:n { \l_b_int / 100 } }
```

이것을

```
\clist_set:Nn \l_cm_clist { r, g, b }
\int_step_inline:nn { \clist_count:N \l_cm_clist }
{
    \tl_set:cf { l_ \clist_item:Nn \l_cm_clist { ##1 } _ tl }
      { \fp_eval:n { \int_use:c { l_ \clist_item:Nn \l_cm_clist { ##1 } _ int
       ↪  } / 100 } }
}
```

이렇게 쓸 수 있다는 것을 지적해둔다. 항목이 여러 개면 더 나은 방법이 될 때도 있다.

한 글자를 다음과 같이 찍을 수 있으므로

```
\ExplSyntaxOn

\cs_new:Npn \print_char:n #1
{
    \gen_rand_color:
    \colorbox{back}{\color{fore} #1}
}

\print_char:n {H}

\ExplSyntaxOff
```

H

원하는 명령을 작성할 수 있게 되었다.

```
\ExplSyntaxOn
\NewDocumentCommand \colorful { m }
{
    \tl_map_inline:nn { #1 }
    {
        \print_char:n { ##1 }
    }
}

\ExplSyntaxOff

\colorful{Hello}
```

Hello

컴파일할 때마다 색상 배열이 달라질 것이다. 알록달록 예쁘기만 하면 된다.
여기서는 전경색과 배경색을 단순히 100에서 뺀 R, G, B 값을 가지고 만들었지만 실제로는 색상에 대한 이해와 토론이 필요한 주제이다. 이에 대하여 더 다루지 않는다.

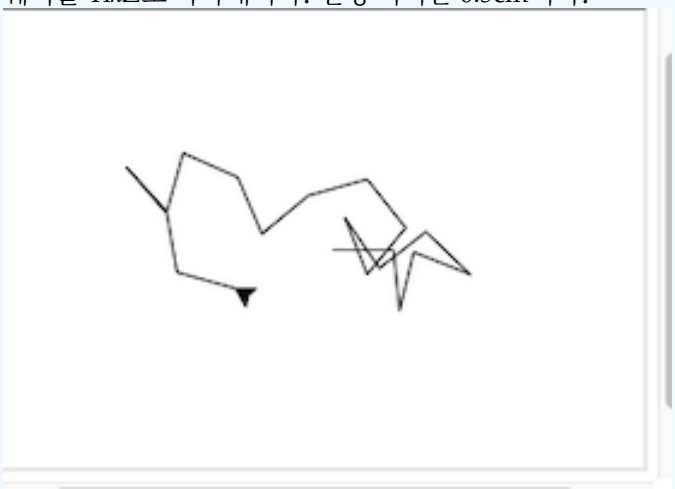
연습문제

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?

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



문제

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

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

정답지는 다음과 같다.

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

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

입력: `\Score{Cheolsu}`
출력: Cheolsu: 20

15 prop 자료형, property list

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

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

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

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

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

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

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

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

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

\bigskip

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

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

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

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

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

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

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

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

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

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

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

채점을 하려면 정답지를 만들어야 한다.

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

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

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

```
\ExplSyntaxOn
\int_new:N \l_score_int

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

        #1:~\int_eval:n { 10 * \l_score_int }
    }
    {
        Something~went~wrong.
    }
}

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

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

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

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

Cheolsu: 20
Yeonghi: 80

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

연습문제

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

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

인데, 이것을

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

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

입력: `\xfacror{24}`
출력: $2^3 \times 3$

문제

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

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

16 keys 자료형

LaTeX Command에 대한 복습 LaTeX에서 "명령"을 만들 적에는

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

expl3 expl3

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

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

xparse xparse xparse

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

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

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

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

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

기본 개념 우리가 하고 싶은 것은, 예를 들어

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

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

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

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

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

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

무슨 일이 일어났는지 다음 테스트 코드로 확인하자.

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

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

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

\ExplSyntaxOff
```

white
blue

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

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

이렇게 쓰는 것을 여기서는

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

로 쓰고 있으므로, 당연히

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

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

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

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

`.code:n`

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

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

이것은

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

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

unknown

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

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

여기까지 배운 것을 바탕으로 주어진 문제를 풀 수 있다.

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

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

\NewDocumentCommand \WordColor { o m }
{
    \IfValueTF { #1 }
    {
        \keys_set:nn { wordcolor } { #1 }
    }
    {
        \keys_set:nn { wordcolor }
        {
```

```
            bgcolor = white,
            fgcolor = blue,
            fsize = normalsize
        }
    }

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

\ExplSyntaxOff

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

test test test

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

연습문제

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

keys 데이터타입에는 이밖에도 key를 하위 그룹으로 분할한다든가 하는 복잡한 조작을 가능하게 하고 있지만 여기서는 더 다루지 않겠다. 다만 choices에 관한 예를 하나 들어두고 마치기로 한다. 둘 이상의 선택도 가능하게 하는 `.multichoices`의 예도 interface3 문서에 나와 있다.

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

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

간단히 테스트해보자.

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

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

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

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

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

연습문제

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

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

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

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

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

연습문제

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

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

연습문제

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

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

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

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

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

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

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

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

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

17 \use:c

예제

다섯 개의 필수 인자를 받는 명령 \nothing이 있다. 인자가 들어오는 순서대로 \l_a_tl, \l_b_tl, \l_c_tl, \l_d_tl, \l_e_tl로 반환하는 일을 하고 그친다.

한 개의 매크로를 가리키는 인자지시자는 :N이다. 예를 들어

```
\foo:N \l_tmpa_tl
```

이라는 것은 \foo:N이라는 함수(cs)가 \l_tmpa_tl이라는 매크로를 인자로서 취한다는 의미이다.
그런데 가끔은 "매크로"를 가리키기 위해서 그 "이름"만을 제공해야 할 때가 있다. 즉

```
\l_tmpa_tl
{l_tmpa_tl}
```

위의 것은 "매크로"이고 아래의 것은 "매르코의 이름"만을 가진 것이다. 이 매크로의 이름을 "csname"이라고 하고 이를 나타내기 위한 인자지시자가 c이다.
그러므로 \foo:N의 변형인 \foo:c가 있다고 하면

```
\foo:N \l_tmpa_tl
\foo:c {l_tmpa_tl}
```

이 둘은 동일한 의미이다. plain TEX에서 다음 둘이 동일한 것과 같다.

```
\mymacro
\csname mymacro\endcsname
```

이런 것이 언제 필요한가? 예를 들어보자면 다른 명령을 정의하기 위한 명령 \jt가 있다고 하자. 이 명령은 letter 인자를 받아서 그 앞에 \cnt@를 붙여준다고 하자.

```
\jt{home}{20}
```

이 명령의 수행 결과가

```
\def\cnt@home{20}
```

과 같이 되게 하려면 어떻게 해야 하는가?

```
\makeatletter
\def\jt#1#2{\expandafter\def\csname cnt@#1\endcsname{#2}}
\jt{home}{20}
\cnt@home
\makeatother
```

20

위의 코드에서 \expandafter가 꼭 필요한데, \def\csname이란 문장은 \csname이라는 매크로를 정의하려 드는 것이라서 허용되지 않기 때문이다.
expl3는 간단히 다음처럼 해도 좋다.

```
\ExplSyntaxOn
\cs_new:Npn \jt_exam:nn #1 #2
{
    \tl_set:cn { cnt_#1_tl } { #2 }
}

\jt_exam:nn { home } { 20 }
\tl_use:N \cnt_home_tl
\ExplSyntaxOff
```

20

\tl_use:N 대신 \tl_use:c를 쓰면

```
\ExplSyntaxOn
\jt_exam:nn { home } { HOME }
\tl_use:c { cnt_home_tl }
\ExplSyntaxOff
```

HOME

실제로 LaTeX에서 \thesection, \thesubsection과 같은 장절번호 명령, \c@page, \c@chapter와 같은 카운터 명령을 설정하는 것은 모두 위와 비슷한 방법으로 이루어진다.
또한, \tl_set: 명령은 상응하는 \tl_new:N가 없어도 새로운 변수에 값을 할당하는 것이 허용되는데 만약 이것이 되지 않는다면 위와 같은 코드를 쓸 때 엄청나게 불편할 것이다.
TMI이 아닌가 싶지만, LaTeX에서

```
\newcounter{abc}
```

명령을 실행하면 \c@abc라는 카운터 변수가 하나 생겨난다. 다음 코드를 보아라. 같은 결과를 내는 세 가지 방법이 나열되어 있다.

```
\makeatletter
\ExplSyntaxOn
\newcounter{abc}\setcounter{abc}{101}
\int_use:N \c@abc, \the\c@abc, \theabc
\ExplSyntaxOff
\makeatother
```

101,101,101

plain TeX에서

```
\newcount\a
```

라고 하면 \a를 바로 카운터로 쓸 수 있는데 비해 LaTeX에서

```
\newcounter{abc}
```

라고 하면 이 LaTeX 카운터의 조작은 \stepcounter, \setcounter, \addtocounter로 하고 그 결과를 \the...로 표현한다는 것을 우리는 이미 잘 알고 있다.

```
plaintex:
  \newcount\a \a=10 \advance\a by10 \the\a

e-tex:
  \newcount\b \b=\numexpr 10+20\relax \the\b

latex:
  \newcounter{a}%
  \setcounter{a}{10}\stepcounter{a}%
  \addtocounter{a}{10}%
  \thea

expl3:
\ExplSyntaxOn
  \int_new:N \l_a_int
  \int_set:Nn \l_a_int { 10 + 10 }
  \int_incr:N \l_a_int
  \int_use:N \l_a_int
\ExplSyntaxOff
```

plaintex: 20
e-tex: 30
latex: 21
expl3: 21

예제를 풀어보자. 1, 2, 3 등은 숫자(other)라서 변수 이름에 쓸 수 없으므로 이를 alph로 바꾸어서 넣어야 한다.

```
\ExplSyntaxOn
\NewDocumentCommand \nothing { mmmmm }
{
    \seq_set_from_clist:Nn \l_tmpa_seq { #1, #2, #3, #4, #5 }

    \seq_map_indexed_inline:Nn \l_tmpa_seq
    {
        \tl_set:cn { l_ \int_to_alph:n { ##1 } _tl } { ##2 }
    }
}

\nothing{Tomato}{Potato}{Banana}{Apple}{Kiwi}

\tl_use:N \l_a_tl,~\tl_use:N \l_d_tl
\ExplSyntaxOff
```

Tomato, Apple

18 weird 인자형

예제

`\testcmd{10.25}`와 같이 주어졌을 때 소수점을 기준으로 10과 25를 분리하고 각각 별도의 매크로(또는 tl)에 넣어 반환하여라.

esgutil 패키지에서 `\esg_fp_format:nn` 명령을 구현하기 위해서는 주어진 문제와 같은 일을 해야 했다. 이에 대해 간략히 알아본다. 문제를 간단히 하기 위해 들어오는 인자에 도트(.)가 없는 경우는 일단 고려하지

않기로 한다.
이 문제에 대한 전통적 해법은 다음과 같다.

```
\makeatletter
\def\splitbydot#1{\expandafter\@split@by@dot#1\end}
\def\@split@by@dot#1.#2\end{\def\a{#1}\def\b{#2}}
\makeatother
\splitbydot{10.25}
A = \a \\
B = \b
```

A = 10
B = 25

위의 예에서 `\@split@by@dot`과 같이 사용자 인터페이스와 무관하고 내부적으로 사용되는 함수에 @문자를 섞어쓰는 관행이 LaTeX에 있다. @문자의 catcode는 letter가 아니라 other이므로 원래는 매크로 이름에 쓸 수 없는데 이를 프로그래밍을 위하여 (임시로) letter로 만들어 매크로 이름의 일부로 쓸 수 있게 하라는 선언이 `\makeatletter`이다. @문자를 원래의 other로 되돌리라는 것은 `\makeatother`이다. 이 범위 안에서는 @을 letter로서 매크로 이름에 쓸 수 있으며, LaTeX style package인 `.sty`는 별도로 선언하지 않아도 `\makeatletter` 상태가 된다. (이 작용은 `\usepackage` 명령이 하는 일이므로 만약 단순히 `\input{foo.sty}`하였다면 그리 되지 않을 것이며, 만약 `.sty` 파일 내부에서 `\makeatother`를 선언하였다면 그 후로는 @이 other가 된다. 스타일 패키지를 만들 때 주의하여야 하는 사항들이다.)

expl3의 입장에서 생각하면 이것은 굉장히 간단한 문제이다. 바로 생각나는 해법은 다음과 같은 것이다.

```
\ExplSyntaxOn
\tl_new:N \l_a_tl
\tl_new:N \l_b_tl
\cs_new:Npn \split_by_dot:n #1
{
    \tl_set:Nn \l_a_tl { #1 }
    \regex_replace_once:nnN { ^ (.+?) \. (.*?) $ } { \1 } \l_a_tl
    \tl_set:Nn \l_b_tl { #1 }
    \regex_replace_once:nnN { ^ (.+?) \. (.*?) $ } { \2 } \l_b_tl
    a ~=~ \tl_use:N \l_a_tl \\
    b ~=~ \tl_use:N \l_b_tl
}
\split_by_dot:n { 10.279 }
\ExplSyntaxOff
```

a = 10
b = 279

실로 명쾌하다고 하겠다.
들어온 수를 "수"로 보고 계산하는 것도 간단하다.

```
\ExplSyntaxOn
\cs_new:Npn \split_by_dot_num:n #1
{
    \fp_set:Nn \l_tmpa_fp { #1 }
    \fp_set:Nn \l_tmpb_fp { trunc ( \l_tmpa_fp ) }
    \tl_set:No \l_a_tl { \fp_use:N \l_tmpb_fp }
    \tl_set:No \l_b_tl { \fp_eval:n { \l_tmpa_fp - \l_tmpb_fp } }
    a ~=~ \tl_use:N \l_a_tl \\
    b ~=~ \tl_use:N \l_b_tl
```

```
}

\split_by_dot_num:n { 10.79 }
\ExplSyntaxOff
```

a = 10
b = 0.79

이 방법은 소수 부분이 정말로 소수로 표현되기 때문에 특정한 용도에 더 적합할 수 있다.
마지막으로 처음에 보인 plain TEX 코드를 expl3로 직접 번역하면 다음처럼 된다.

```
\ExplSyntaxOn
\cs_new:Npn \split_by_dot_exp:n #1
{
    \split_by_dot_exp_subfn:w #1\q_stop
    a ~=~ \tl_use:N \l_a_tl \\
    b ~=~ \tl_use:N \l_b_tl
}

\cs_new:Npn \split_by_dot_exp_subfn:w #1 . #2 \q_stop
{
    \tl_set:Nn \l_a_tl { #1 }
    \tl_set:Nn \l_b_tl { #2 }
}

\split_by_dot_exp:n { 10.297 }

\ExplSyntaxOff
```

a = 10
b = 297

이 기법에 사용된 것은 w (weird) arguments를 이용하는 것이다. 특별한 인자 형식으로부터 인자를 넘겨받기 위한 목적으로 사용한다. 종지 위치를 \q_stop이라는 quark으로 표시하는 것은 아까 보인 코드에서 \end를 주는 것과 동일하다.

19 인자 수

예제

12개의 필수(mandatory) 인자를 받아서 처리하는 명령 \testtwelve를 작성해보아라.

\fbox하고 콤마로 나열하는 걸 해보겠다.
#10이란 게 없기 때문에 9개까지만 인자를 설정할 수 있다. 그 이상이라면 그 이후의 인자를 취하는 명령을 해당 명령 마지막에 두어서 처리하는 것이 가능하다.

```
\ExplSyntaxOn
\NewDocumentCommand \testtwelve { mmmmmm }
{
    \fbox{#1}, \fbox{#2}, \fbox{#3}, \fbox{#4},
    \fbox{#5}, \fbox{#6},
    \testtwelvenext
}
```

```
\NewDocumentCommand \testtwelvenext { mmmmmm }
{
    \fbox{#1}, \fbox{#2}, \fbox{#3}, \fbox{#4},
    \fbox{#5}, \fbox{#6},
}
\ExplSyntaxOff

\testtwelve{a}{b}{c}{d}{e}{f}{g}{h}{i}{j}{k}{l}
```

a, b, c, d, e, f, g, h, i, j, k, l,

expl3의 함수(cs)도 필요하다면 같은 방식으로 할 수 있다.
그런데 가만히 생각해보면 인자를 취하는 것은 전형적인 "재귀" 과정이다. 그러니까,

```
\ExplSyntaxOn
\int_zero:N \l_tmpa_int
\cs_new:Npn \test_twelve:n #1
{
    \fbox{#1}
    \int_incr:N \l_tmpa_int
    \int_compare:nTF { \l_tmpa_int < 12 }
    {
        , \test_twelve:n
    }
    {
        .
    }
}

\test_twelve:n {a}{b}{c}{d}{e}{f}{g}{h}{i}{j}{k}{l}
\ExplSyntaxOff
```

a, b, c, d, e, f, g, h, i, j, k, l.

열두 개의 인자를 취하는 것은 위와 같다.

20 가변개의 인자

예제

새로이 정의하는 명령은 다음과 같은 형식이다.

```
\mycmd{1}{2}{5}{2}
```

그러면 인자로 주어지는 수만큼 점을 찍어준다.

• •• ••••• ••
인자는 몇 개가 주어질지 사전에 알 수 없다. 단 중괄호가 아닌 부호가 오면 그 후로는 인자로 보지 않는다.

plain TEX으로는 다음처럼 할 수 있다. 테스트를 위하여 마지막에 (4)와 같이 중괄호가 아닌 부호를 두었다.

```
\newcount\n
\def\mycmd{\futurelet\next\mycmdsub}
\def\mycmdsub{\ifx\next\bgroup\expandafter\mycmdproc\fi}
\def\mycmdproc#1{\n=1\loop\textbullet\advance\n
 ↪  by1\ifnum\n<#1\repeat\quad\mycmd}

\mycmd{2}{5}{3}(4)
```

•• ••••• ••• (4)

이 코드는 설명하지 않겠다.
이런 종류의 "재귀" 코드에서 가장 중요한 것은 종료 조건을 분명하게 하는 것이다. 예를 들어

```
\mycmd{{1}{2}{5}{3}}
```

과 같이 인자 범위를 명확히 설정해주거나

```
\mycmd{1}{2}{5}{3};
```

과 같이 종지부호를 적게 하는 방식으로 명령을 정의하면 그 인자를 처리하는 것이 아주 쉬워진다. 두 번째와 같이 종지부호를 세미콜론으로 하는 경우의 코드를 보자.

```
\ExplSyntaxOn
\NewDocumentCommand \mycmd {u;}
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_map_inline:Nn \l_tmpa_tl
    {
        \mycmd_dot:n { ##1 }
    }
}

\cs_new:Npn \mycmd_dot:n #1
{
    \int_step_inline:nn { #1 }
    {
        \textbullet
    }
    \quad
}
\ExplSyntaxOff

\mycmd{1}{2}{5}{3};
```

• •• ••••• •••

실용적 목적으로 문서 명령을 작성하는 경우라면 되도록 "명시적 종료 토큰"을 사용자가 입력하게 만드는 것이 좋다.
그러나 어찌됐든 처음에 보였듯이 종지부호 같은 것이 없이 가변 인자를 구현해야 한다면, 어쩔 수 없이 토큰 검사를 하는 도리밖에 없다. 처음에 보인 plain TEX의 코드를 expl3 방식으로 다시 쓰면 다음과 같을 것이다. 마지막의 (5)는 테스트를 위하여 중괄호가 아닌 부호를 둔 것이다.

```
\ExplSyntaxOn
\cs_new:Npn \my_cmd:n #1
{
    \mycmd_dot:n { #1 }
```

```
    \peek_catcode:NT \c_group_begin_token
    {
        \my_cmd:n
    }
}

\NewDocumentCommand \mycmd { }
{
    \my_cmd:n
}
\ExplSyntaxOff

\mycmd{3}{2}{1} {4}
```

●●● ●● ● 4

이 명령에서 유심히 볼 것은 중괄호가 아닌 것이 오면 끝난다고 했으므로 space token이 와도 마찬가지로 종료된다는 것이다. 만약 space를 무시하고 위와 같은 경우에 마지막 {4}도 유효하게 만들려면 \peek_catcode_ignore_spaces:N을 쓰면 되기는 하지만 space로 종료하지 못하면 다음 non-space token이 중괄호가 아니어야 하므로, 다른 식의 종료 조건을 반드시 구비하여야 한다. 그렇게 하지 않으면 에러가 발생할 가능성이 너무 높아진다.

문서 명령을 정의하면서 인자 자리를 비우고 재귀 함수인 \my_cmd:n을 부르는 것으로 그친 것은 드물지 않은 기법이므로 익숙할 것이다. 재귀함수로써 그 이후에 이어지는 입력 스트림을 처리하는 명령을 작성할 때는 재귀함수의 호출이 이 명령의 "마지막" 토큰이어야 한다는 점을 기억하자. 명시적으로 주어지는 인자는 없지만 \my_cmd:n이 어차피 한 개의 토큰을 받아서(absorb) 처리하기 시작할 것이므로 결과적으로 이후의 인자를 모두 처리하고 종료할 것을 예상할 수 있다. 요컨대, \futurelet으로 다음에 올 토큰을 검사하던 것을 expl3에서는 \peek_catcode:나 \peek_charcode:를 통해 대부분 해결할 수 있다. (다만 다음에 올 토큰이 space token일 경우에 예외적인 취급이 필요할 수 있는데 이에 대해서는 이후 절을 달리하여 토론하기로 한다.)

지금까지 배운 내용으로 다음 문제를 해결할 수 있다.

연습문제

다음과 같은 사용자 Ti*k*Z 명령을 작성하려 한다. \mydraw(0,0)(1,1)(2,1)라고 하면

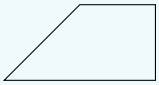
```
\draw (0,0)--(1,1)--(2,1)--cycle;
```

로 그려주는 명령이다. 명령의 마지막에 세미콜론이 붙지 않으며 가변 개의 점을 지정할 수 있다.

Special Course는 연습문제를 숙제로 삼지 않으므로 모범답안을 보이면 다음과 같다.

```
\ExplSyntaxOn
\NewDocumentCommand \mydraw {}
{
    \tl_clear:N \l_tmpa_tl
    \my_draw:w
}

\cs_new:Npn \my_draw:w (#1,#2)
{
    \tl_put_right:Nn \l_tmpa_tl { (#1,#2) }
    \peek_charcode:NTF (   %%% if next token is open-paren
    {
```

```
        \tl_put_right:Nn \l_tmpa_tl { -- }
        \my_draw:w
    }
    {
        \tl_put_right:Nn \l_tmpa_tl { -- cycle }
        \draw_final:N \l_tmpa_tl
    }
}

\cs_new:Npn \draw_final:N #1
{
    \exp_last_unbraced:No
    \draw #1 ;
}
\ExplSyntaxOff
\begin{tikzpicture}
\mydraw(0,0)(1,1)(2,1)(2,0)
\end{tikzpicture}
```



\my_draw:w의 인자형에 대해 조금 해설을 붙이자면 실제로는 \my_draw:w (#1)과 같이 하고 #1만을 put right해주어도 이 코드에서는 문제를 일으키지 않는다. 그런데 만약 x좌표와 y좌표를 조작해야 할 일이 있다면 저런 형식으로 인자를 받는 것이 좋다. 또한 쉼표가 들어가지 않은 material이 괄호 안에 들어온다고 다 유효한 것으로 취급하는 것보다는 쉼표가 없으면 아예 에러를 토하는 것이 낫다. 예시 코드가 그런 작용을 한다.

21 modulo 연산과 조건식

예제

\int_compare:nTF { \int_mod:nn { #1 } { 5 } == 0 } {T}{F}과 같이 쓰는 조건식을 \my_mod:TF (#1 / 5) {T}{F}와 비슷하게 줄여쓸 수 없을까? 그런 명령을 정의해보아라.

expl3에 modulo 연산자가 없고 함수만 있기 때문에 함수명을 길게 적는 것이 귀찮은 일일 수 있다.

```
\ExplSyntaxOn

\prg_new_conditional:Npnn \my_modzero:w (#1//#2) { p, T, F, TF }
{
    \int_compare:nTF { \int_mod:nn { #1 } { #2 } == 0 }
    {
        \prg_return_true:
    }
    {
        \prg_return_false:
    }
}

\my_modzero:wTF (15 // 5) { TRUE } { FALSE } \\
\my_modzero:wTF (16 // 5) { TRUE } { FALSE } \\
\bool_if:nTF { \my_modzero_p:w (17//5) } { T } { F }
```

```
\ExplSyntaxOff
```

TRUE
FALSE
F

인자형을 :nn으로 한다면 두 개의 중괄호 인자가 오는 것으로 약속했고, 이 함수는 (#1//#2) 형식, 즉 중괄호를 쓰지 않고 괄호(parenthesis)와 슬래시로 인자를 주기 때문에 인자형을 :w로 해두는 편이 낫다. 물론 #1이나 #2 위치에 중괄호로 둘러싸인 정수 표현식이 오는 것은 상관없다. (다음 보기는 테스트용에 가깝다. 너무 복잡한 식이 오면 나중에 알아보기 어려워진다.)

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int { \fp_eval:n { round ( sqrt ( 200 ) ) } }
\my_modzero:wTF ( { \l_tmpa_int * ( 3 + \int_max:nn { 5 } { 9 } ) } // 3 )
{ divisible~by~3 }
{ not~divisible~by~3 }
\ExplSyntaxOff
```

divisible by 3

\my_modzero_p:w는 boolean 값을 반환한다. \my_modzero:wTF는 (T 또는 F중 하나는 생략될 수 있음) "주어진 나머지 연산식이 0이면" 이라는 조건에 따라 실행할 코드를 줄 수 있다.

22 tl map과 space token

예제

입력되는 문자열의 각 문자에 fbox를 치는 명령을 작성하되 space 위치에 가로 4pt, 세로 7pt의 막대를 그려라.

입력: \test{Lorem ipsum dolor}
출력: L o r e m i p s u m d o l o r

tl에서 space는 보존된다. (...ignore_spaces 관련 함수를 쓰지 않은 한.)

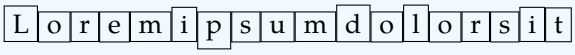
```
\ExplSyntaxOn
\NewDocumentCommand \foo { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \l_tmpa_tl
}
\ExplSyntaxOff

\foo{Lorem ipsum dolor sit}
```

Lorem ipsum dolor sit

그러나, \tl_map_...에서 next token은 non-space token을 가리킨다.

```
\ExplSyntaxOn
\NewDocumentCommand \foo {m }
```

```
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_map_inline:Nn \l_tmpa_tl { \fbox { ##1 } }
}
\ExplSyntaxOff

\foo{Lorem ipsum dolor sit}
```



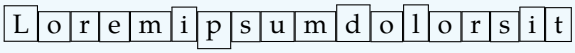
이와 같은 일은 \peek_catcode:N \c_space_token에서도 일어난다. 단순 입력 스트림에서는 스페이스 토큰을 peeking하는 것이 가능하기 때문에 다음과 같은 코드는 동작하지만 (이와 같이 중괄호로 묶어서 인자로 전달하지 않고 토큰을 늘어놓는 방식은 앞서 소개한 가변 인자 처리와 사실상 동일한 아이디어임을 상기하여라.)

```
\ExplSyntaxOn
\cs_new:Npn \test_r:n #1
{
    \peek_catcode:NTF \c_space_token
    {
        #1 \rule{4pt}{7pt}
    }
    {
        \fbox{#1}
        \test_r:n
    }
}

\cs_set_eq:NN \test \test_r:n
\ExplSyntaxOff

\test Lorem ipsum dolor sit
```

\NewDocumentCommand의 m인자 안에 들어가고 나면 space를 peek할 수 없게 된다. (\peek_... 함수는 다른 함수의 인자로 들어가지 않은 입력 스트림에서 정상적으로 동작한다는 것을 기억하자.) 아래 코드에서 \peek_catcode 부분은 항상 false로서 스페이스 검출에 성공하지 못하였다.

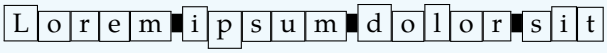
```
\ExplSyntaxOn
\NewDocumentCommand \foo { m }
{
    \foo_recur:n #1 \q_recursion_tail \q_recursion_stop
}

\cs_new:Npn \foo_recur:n #1
{
    \quark_if_recursion_tail_stop:n { #1 }
    \peek_catcode:NTF \c_space_token
    {
        #1 \rule{4pt}{7pt}
    }
    {
        \fbox{#1}
    }
```

```
    \foo_recur:n
}

\ExplSyntaxOff
\foo{Lorem ipsum dolor sit}
```



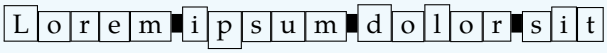
주어진 문제를 풀기 위해서 space를 처리하는 방법은 몇 가지가 있다. space 단위로 seq에 넣어 처리하는 것이 가장 직관적이고

```
\ExplSyntaxOn
\NewDocumentCommand \foo { m }
{
    \seq_set_split:Nnn \l_tmpa_seq { ~ } { #1 }
    \seq_map_indexed_inline:Nn \l_tmpa_seq
    {
        \f_each:n { ##2 }
        \int_compare:nT { ##1 < \seq_count:N \l_tmpa_seq }
        { \rule{4pt}{7pt} }
    }
}

\cs_new:Npn \f_each:n #1
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_map_inline:Nn \l_tmpa_tl { \fbox { ##1 } }
}
\ExplSyntaxOff

\foo{Lorem ipsum dolor sit}
```



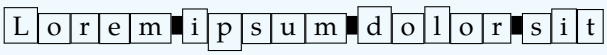
다음처럼 해볼 수도 있다. space를 특정 부호(여기서는 |)로 미리 바꾸어두는 것이다.

```
\ExplSyntaxOn
\NewDocumentCommand \foo { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \regex_replace_all:nnN { \s } { | } \l_tmpa_tl
    \tl_map_inline:Nn \l_tmpa_tl
    {
        \str_if_eq:nnTF { ##1 } { | }
        {
            \rule{4pt}{7pt}
        }
        {
            \fbox{##1}
        }
    }
}

\ExplSyntaxOff
\foo{Lorem ipsum dolor sit}
```



또는 "재귀"적으로 정의하는 방법도 있다. `\q_recursion_tail (stop)` 기법은 한 번 써보았으니 여기서는 `\q_nil`을 끝에 두고 종료 조건을 설정하는 방식으로 해보았다.

```
\ExplSyntaxOn
\NewDocumentCommand \foo { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \regex_replace_all:nnN { \s } { | } \l_tmpa_tl
    \exp_last_unbraced:Nx
    \foo_fn:n \l_tmpa_tl \q_nil
}

\cs_new:Npn \foo_fn:n #1
{
    \quark_if_nil:NF #1
    {
        \token_if_eq_charcode:NNTF #1 |
        {
            \rule{4pt}{7pt}
        }
        {
            \fbox{#1}
        }
        \foo_fn:n
    }
}

\ExplSyntaxOff
\foo{Lorem ipsum dolor sit}
```



이 코드는 동작하지만 `\quark_if_...`로 처리하는 편이 더 안전하다. 연습 삼아 `\quark_if_recursion_...`으로 똑같은 코드를 적어보겠다.

```
\ExplSyntaxOn
\NewDocumentCommand \foo { m }
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \regex_replace_all:nnN { \s } { | } \l_tmpa_tl
    \exp_last_unbraced:Nx
    \foo_s:n \l_tmpa_tl \q_recursion_tail \q_recursion_stop
}

\cs_new:Npn \foo_s:n #1
{
    \quark_if_recursion_tail_stop:n { #1 }

    \token_if_eq_charcode:NNTF #1 |
    {
        \rule{4pt}{7pt}
    }
    {
        \fbox{#1}
```

```
    }
    \foo_s:n
}

\ExplSyntaxOff
\foo{Lorem ipsum dolor sit}
```

Lorem ipsum dolor sit

마지막으로 다음 코드는 peek catcode로 space token을 검출하여 뭔가를 할 수 있는 "재귀" 함수 코드이다. 여기서는 space token 직전 토큰의 색상을 빨갛게 해보았다. [FS] (Full Stop) 부호는 종료 조건(스페이스가 더이상 종료 조건이 되지 못하기 때문에 마침표 사용)을 검출하기 위한 테스트 코드이다.

```
\ExplSyntaxOn

\cs_new:Npn \_chk_and_exit:nN #1 #2
{
    \token_if_eq_catcode:NNF #1 .
    { #2 }
}

\cs_new:Npn \foo_rc:n #1
{
    \str_if_eq:nnTF { #1 } { . }
    { [FS] }
    {
        \peek_catcode:NTF \c_space_token
        {
            { %%% \fboxsep made local
                \dim_set:Nn \fboxsep { 1pt }
                \colorbox{gray!30}{\color{red}\fbox{#1}}
            }
            \rule{4pt}{7pt}
            \_chk_and_exit:nN #1 \foo_rc:n
        }
        {
            \fbox{#1}
            \_chk_and_exit:nN #1 \foo_rc:n
        }
    }
}

\cs_set_eq:NN \foo \foo_rc:n

\ExplSyntaxOff
\foo Lorem ipsum dolor sit. Hello boys.
```

Lorem ipsum dolor sit [FS] Hello boys.

plain TEX에서는 \@sptoken만 가지고 next token으로 검출하는 것이 비교적 간단하다. 그런데 expl3에서 살짝 복잡해진 이유는 expl3가 space token을 건드려놓았기 때문이다. \ExplSyntaxOn으로 스페이스는 모두 사라진다는 걸 생각해보면 expl3가 space token을 특별하게 취급하고 있다는 것이 짐작될 것이다. 그리고 ...ignore_spaces 관련 함수도 많고.
그 덕분에 예전에는 할 수 없었던 여러 가지를 에러 없이 손쉽게 하게 된 것도 있고, 이처럼 space 자체를 제어하려면 조금 고민이 필요한 부분도 생겨나고 한 것이라고 보면 되겠다. 전체적으로 보아서 expl3의

스페이스 취급은 실보다는 득이 많은 방향이었다는 판단이므로 여기 소개한 기법을 필요할 때 활용할 수 있으면 그것으로 좋다.

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

예제

아래 "문제상황"을 읽고 적절한 조언을 작성하여라.

문제상황:
철수는 방학 아르바이트로 학습지 조판을 맡게 되었다. 이 학습지의 중간중간에 나오는 "보기" 표시를 위해서 다음과 같은 명령을 만들었다.

```
\ExplSyntaxOn
\box_new:N \l_boghi_box
\hbox_set:Nn \l_boghi_box
{
    \tikz \node [
        drop~shadow = {opacity=.35},
        rounded~corners=1pt,
        fill=orange!30, draw=blue!80,
        thin, inner~sep=3pt
    ] { \sffamily\bfseries\scriptsize 보기 };
}

\box_use:N \l_boghi_box
\ExplSyntaxOff
```

보기

모양은 원하는 대로 잘 된 거 같아서 만족하면서 이 "보기 박스"를 식자하는 문서 명령을 다음과 같이 작성하였다. (이를 위하여 preamble에 `\usetikzlibrary{shadows}`를 추가하여야 했다.)

```
\ExplSyntaxOn
\NewDocumentCommand \bogi { }
{
    \box_use:N \l_boghi_box
}
\ExplSyntaxOff
```

그런데 두 가지 문제가 생겼다. 이의 해결방법이 무엇인가?
문제상황 1. "보기" 박스가 다른 텍스트들에 비하여 위치가 잘 맞지 않는다.

답안지를 다음 **\bogi** 와 같이 작성하세요.

답안지를 다음 보기 와 같이 작성하세요.

문제상황 2. 박스가 엉뚱한 위치에 가 있을 때가 있다.

\bogi 에 표시된 것을 보세요.

보기
에 표시된 것을 보세요.

23 boxes

TEX은 모든 것을 box로 식자한다. 사용자 입장에서 매번 박스의 위치와 구조를 모두 알고 있어야 할 필요는 없지만 최소한의 지식은 필요하다. expl3 역시 box를 다루는 함수를 (충분히) 제공하고 있다.
한편, TEX은 박스를 조판할 때 mode라는 것에 의존한다. 기본적인 mode로는 horizontal mode, vertical mode, math mode 세 가지가 있고 수평모드와 수직모드는 제한 수평모드(restricted horizontal mode)나 내부 수직모드 등으로 그 상황이 세분된다. 이에 대한 자세한 사항을 설명하는 것은 이 강좌의 중심주제가 아니다. 그렇지만 박스가 문제가 되면 이 박스가 놓이는 위치가 수평모드 상황인지 수직모드 상황인지를 구별하여야 할 필요가 있다는 사실을 지정해둔다.
박스는 다시 hbox와 vbox로 구분된다. 이것은 조판 상황을 가리키는 hmode나 vmode와는 달리 박스 안에 오는 내용물을 어떻게 조판하는가 하는 문제에 관련이 있다. 이에 대한 자세한 이론을 소개하는 것 역시 이 강좌의 목적이 아니므로 더 상세한 내용은 *The TEXbook*이나 기타 TEX 참고서를 보는 것이 좋겠다. 우리에게 필요한 지식은 다음과 같다.

가로박스(hbox) 수직적 조판—행자르기, 수직 간격주기 등—동작이 전혀 일어나지 않으며 내용물이 가로로 놓인다. 이 박스의 폭(width)은 기본적으로 내용물을 조판한 결과에 따르지만(이것을 natural width라고 한다) 임의의 다른 값을 부여할 수도 있다. `\mbox`가 가장 대표적인 가로박스를 만드는 명령이다.

세로박스(vbox) 정해져 있는 폭(width)에 맞추어서 내용물을 행자르기나 문단나누기를 실행하면서 조판한 결과를 가지고 있는 박스이다. 폭은 고정이고 높이와 깊이는 내용물이 모두 조판된 후에 결정되는 것이 디폴트이나 임의의 값을 부여할 수 있다. 첫 줄을 baseline에 놓고 아래로 길어지는 vbox가 `\vtop`이며 마지막 줄이 baseline에 놓이는 vbox가 `\vbox`이다. TEX의 입장에서는 "페이지"라는 것 자체가 하나의 특별한 vbox이다.

아주 소박하게 hbox와 vbox의 예를 들고 가자.

```
\begin{minipage}{5cm}
Lorem ipsum dolor sit amet,
\hbox{consectetur adipiscing elit, sed do eiusmod tempor incididunt}
ut labore et dolore magna aliqua.
\end{minipage}
```

Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod tempor incididunt
ut labore et dolore magna ali-
qua.

```
Lorem
\vtop{\hsize=3cm 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
```

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

이후, box primitive는 거의 쓰지 않을 것이다. 각각 거기에 해당하는 expl3 문법이 있으므로 그것을 활용하겠다.
hbox에 대해서는 그 박스의 폭(width)을 (사후적으로) 정해줄 수 있다. 실제로 그 박스 안에 들어 있는 내용물보다 작은 폭을 지정하면 무슨 일이 일어나는지 보자.

```
\ExplSyntaxOn
\hbox_set:Nn \l_tmpa_box { ipsum~dolor~sit~amet, }
\box_set_wd:Nn \l_tmpa_box { 1cm }
Lorem~\box_use:N \l_tmpa_box
consectetur~adipiscing~elit,~
sed~do~eiusmod~tempor~incididunt~ut~labore~et~dolore~magna~aliqua
\ExplSyntaxOff
```

Lorem ipsumconsectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua

hbox를 set하면서 아예 그 박스 폭을 정해주는 간편한 명령도 있다. 위의 두 줄을 하나의 명령으로 합친 것으로 이해해도 좋다.

```
\ExplSyntaxOn
\hbox_set_to_wd:Nnn \l_tmpa_box { 1cm } { ipsum~dolor~sit~amet, }
Lorem~\box_use:N \l_tmpa_box
consectetur~adipiscing~elit,~
sed~do~eiusmod~tempor~incididunt~ut~labore~et~dolore~magna~aliqua
\ExplSyntaxOff
```

Lorem ipsumconsectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua

vbox는 높이(height)와 깊이(depth)를 (사후적으로) 정해줄 수 있다.
한편, 다음과 같은 것은 어떤 방식으로 식자될지 짐작해보자.

```
\ExplSyntaxOn
\hbox_set:Nn \l_tmpa_box { abcd }
\box_set_wd:Nn \l_tmpa_box { \c_zero_dim }
\vbox_set:Nn \l_tmpb_box { \rule{1cm}{.8cm} }
\box_set_wd:Nn \l_tmpb_box { \c_zero_dim }
\box_set_ht:Nn \l_tmpb_box { \c_zero_dim }
Lorem~\box_use:N \l_tmpa_box ipsum~dolor~\box_use:N \l_tmpb_box sit~amet,
\ExplSyntaxOff
```

Lorem abcdm dolor sit amet,

주어진 위치에 폭이 0pt인 가로박스를 바로 식자할 수 있게 하는 `\hbox_to_zero:n`과 높이가 0pt인 세로박스를 바로 식자할 수 있게 하는 `\vbox_to_zero:n`이 있어서 이런 상황에 유용하게 쓸 수 있다.

박스의 이동 box를 아래나 위로, 왼쪽이나 오른쪽으로 이동하여 식자하게 할 수 있다. up/down은 hmode에서 left/right는 vmode에서만 동작한다.

```
\ExplSyntaxOn
\hbox_set:Nn \l_tmpa_box { LOREM~IPSUM }
Lorem~ipsum~\box_move_up:nn { 7pt } { \box_use:N \l_tmpa_box }
```

```
dolor~\vadjust{\box_move_right:nn { 2em } { \box_use_drop:N \l_tmpa_box }}
 ↪  sit~amet
\ExplSyntaxOff
```

Lorem ipsum LOREM IPSUM dolor sit amet
LOREM IPSUM

hbox인 \l_tmpa_box를 수평모드에서 move up하였다. move right는 수직모드에서만 가능하기 때문에 잠시 수직모드로 빠져나가는 \vadjust를 사용하여 테스트하였다.
\box_use_drop:N은 \box_use:N과 같지만 use한 후에 박스를 비운다.

문단 첫머리에서는 무슨 일이? 우리가 소스를 입력하면서 문단을 구분짓기 위해 두 개 이상의 빈 줄을 둔다. 둘 이상의 빈 줄은 모두 TEX primitive인 \par로 바뀌기 때문에 앞으로 \par에 대해 말하는 것은 모두 빈 줄 두 개 이상인 상황을 가리키는 것이다.
이 "문단과 문단 사이"가 바로 대표적인 vmode 상태이다. 언제 hmode로 들어가는가?[2] \leavevmode가 수평모드로 들어가라는 대표적 명령이다. 이밖에 \indent를 만나거나 단 한 자라도 문자나 수식이 입력되면 그 때부터 수평모드로 간주한다. 그리고 일부 LATEX 매크로는 만약 그 매크로가 불린 위치가 수직모드이면 수평모드로 전환하라는 명령을 내부에 가지고 있다. (이와 관련하여 최근 LATEX 2018년 12월의 마지막 수정에서 매크로 중 일부(e.g. \smash)에 추가적으로 이런 조치가 이루어졌다.)
그렇기 때문에 문단 첫머리 아무 글자도 입력되지 않은 상태는 수직모드이다. 이 위치에서 box를 식자하면 어떻게 될까? 당연히 vmode이므로 box를 놓고 여전히 vmode를 유지한다. 이게 무슨 말이냐면 hmode로 진입하지 못한다는 의미로서, 그 뒤에 다른 문자가 이어지더라도 아직 수직모드이기 때문에 행이 나누어지는 수직 조판 작용이 일어난다는 의미이다. 그 다음에 한 문자라도 오면 그 때부터 수평모드가 되어 정상적인 식자가 이루어진다. 다음 예를 보아라.

```
natural mode

\hbox{\fbox{Lorem}} ipsum dolor sit
```

natural mode
Lorem
ipsum dolor sit

따라서, 만약 자신이 만든 명령이나 환경이 box를 두는 것이고 이것이 문단 첫머리에 놓일 가능성이 있다면 반드시 수평모드를 강제해두어야 한다.

```
natural mode

\ifvmode\leavevmode\fi\hbox{\fbox{Lorem}} ipsum dolor sit
```

natural mode
Lorem ipsum dolor sit

expl3 언어로는 이렇다.

```
\mode_if_vertical:T { \mode_leave_vertical: }
```

[2] 이 문제에 관련하여 김도현 교수의 훌륭한 발표자료 [2]가 있다. 내용이 조금 어렵지만 읽어볼 가치가 충분하다.

LaTeX 사용법을 배우는 과정에서 자주 듣는 조언 중에 "\bigskip이나 \medskip 같은 명령은 문단이 끝난 후 한 줄 띄고 적으라"는 것이 있는데, 문단이 끝난 직후는 여전히 hmode이고 한 줄을 띄어야 vmode가 되는 것이며 위와 같은 수직 이동 명령은 vmode에서 의도대로 동작하기 때문이라는 것을 알 수 있을 것이다.

문제의 해결 제시된 예제의 "문제상황 1"은 박스가 글줄에 맞추어서 수직이동을 해야 한다. 어느 정도 이동하는 것이 적절할까? 박스 안에 식자되는 텍스트의 baseline과 박스가 둘러싼 sep 길이를 더한 만큼 아래로 끌어내리면 될 듯하다.

```
\ExplSyntaxOn

\RenewDocumentCommand \bogi { }
{
    \hbox_set:Nn \l_tmpa_box { \sffamily\bfseries\scriptsize 보기 }
    \dim_set:Nn \l_tmpa_dim { \box_dp:N \l_tmpa_box }
    \dim_add:Nn \l_tmpa_dim { 3pt }   %% inner sep of tikz
    \dim_add:Nn \l_tmpa_dim { .4pt }  %% width of thin line
    \box_move_down:nn { \l_tmpa_dim } { \box_use:N \l_boghi_box }
}
\ExplSyntaxOff

다음 \bogi 와 같이 식자하세요
```

다음 보기 와 같이 식자하세요

이것은 의도한 대로 된 것이다. 왜냐하면 박스 안의 "보기"라는 글자와 박스 바깥쪽의 글자가 baseline이 일치하기 때문이다. 그러나 아마도 만족스럽지 않을 것이다. 보기 박스에 음영 부분이 있으므로 이만큼을 시각적으로 무시하는 것이 좋을 듯하다. 그러므로 일반 글자의 dp의 2배 정도를 내려 식자하게 하여보면,

```
\ExplSyntaxOn
\RenewDocumentCommand \bogi { }
{
    \hbox_set:Nn \l_tmpa_box { \normalfont\normalsize 학 }
    \dim_set:Nn \l_tmpa_dim { \box_dp:N \l_tmpa_box * 2 }
    \box_set_ht:Nn \l_boghi_box { 0pt }
    \box_move_down:nn { \l_tmpa_dim } { \box_use:N \l_boghi_box } \thinspace
}
\ExplSyntaxOff

다음 \bogi 와 같이 식자하세요
```

다음 보기 와 같이 식자하세요

마지막의 \thinspace는 보기 박스의 음영과 그 다음 글자 사이에 약간의 간격을 준 것이다.

"문제상황 2"는 이미 배운 대로 vmode에서 박스를 식자하고 있기 때문에 생겨난 일임을 알겠다.

```
\ExplSyntaxOn
\RenewDocumentCommand \bogi { }
{
    \mode_if_vertical:T { \mode_leave_vertical: }
    \hbox_set:Nn \l_tmpa_box { \normalfont\normalsize 학 }
    \dim_set:Nn \l_tmpa_dim { \box_dp:N \l_tmpa_box * 2 }
    \box_set_ht:Nn \l_boghi_box { 0pt }
    \box_move_down:nn { \l_tmpa_dim } { \box_use:N \l_boghi_box } \thinspace
```

```
}
\ExplSyntaxOff

\bogi 에 표시된 것을 보세요
```

보기에 표시된 것을 보세요

연습문제

기본 9. 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.
```

연습문제

기본 10. 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.

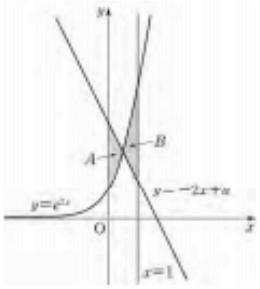
4 수학 영역(가형) 홀수형

11. 실수 전체의 집합에서 미분가능한 두 함수 $f(x)$, $g(x)$가 있다. $f(x)$가 $g(x)$의 역함수이고 $f(1)=2$, $f'(1)=3$이다. 함수 $h(x)=xg(x)$라 할 때, $h'(2)$의 값은? [3점]

① 1 ② $\frac{4}{3}$ ③ $\frac{5}{3}$ ④ 2 ⑤ $\frac{7}{3}$

12. 곡선 $y=e^{2x}$과 y축 및 직선 $y=-2x+a$로 둘러싸인 영역을 A, 곡선 $y=e^{2x}$과 두 직선 $y=-2x+a$, $x=1$로 둘러싸인 영역을 B라 하자. A의 넓이와 B의 넓이가 같을 때, 상수 a의 값은? (단, $1<a<e^2$) [3점]

① $\frac{e^2+1}{2}$ ② $\frac{2e^2+1}{4}$ ③ $\frac{e^2}{2}$
④ $\frac{2e^2-1}{4}$ ⑤ $\frac{e^2-1}{2}$

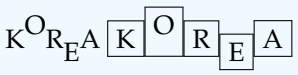


예제

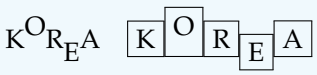
선행학습을 요구한 논문 조진환(2007) [1]의 본문 중에 예제 10과 11의 내용을 잘 읽고 expl3로 구현한다면 어떻게 되겠는지 생각해보아라.

예제 10: hbox의 raise와 lower p. 75의 소스코드는 다음과 같다. \hbox를 raise하거나 lower하는 것이다.

```
K\raise 5pt\hbox{O}R\lower 5pt\hbox{E}A
\fbox{K}\fbox{\raise 5pt\hbox{O}}\fbox{R}\fbox{\lower 5pt\hbox{E}}\fbox{A}
```

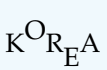


```
\ExplSyntaxOn
K
\hbox_set:Nn \l_tmpa_box { O }
\box_move_up:nn { 5pt } { \box_use:N \l_tmpa_box }
R
\hbox_set:Nn \l_tmpa_box { E }
\box_move_down:nn { 5pt } { \box_use:N \l_tmpa_box }
A

\quad

\fbox{K}
\hbox_set:Nn \l_tmpa_box { O }
\fbox{\box_move_up:nn { 5pt } { \box_use:N \l_tmpa_box }}
\fbox{R}
\hbox_set:Nn \l_tmpa_box { E }
\fbox{\box_move_down:nn { 5pt } { \box_use:N \l_tmpa_box }}
\fbox{A}
\ExplSyntaxOff
```



box 이동 함수 \box_move_. . .는 up, down, left, right 네 종류가 있는데 이 가운데 up, down은 hbox에, left, right는 vbox에 적용한다. 언뜻 보면 \raise와 \lower 같은 primitive를 쓰는 것이 훨씬 간단해보이고 실제로 그런 경우가 많다. 그렇지만 expl3로 함수나 명령을 만들다가 보면 이쪽 문법이 더 이해하기 쉽고 box 변수들을 다루는 데도 편리할 때가 있음을 알게 될 것이다.

```
\ExplSyntaxOn
K
\box_move_up:nn { 5pt } { \hbox:n { O } }
R
\box_move_down:nn { 5pt } { \hbox:n { E } }
A
\ExplSyntaxOn
```

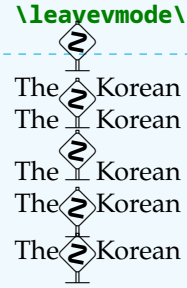


앞선 예에서는 어떤 문자를 \l_tmpa_box에 넣고 이것을 \use_box:N 하는 방식으로 문제를 해결했다. 이것이 box를 사용하는 더 표준적인 방식인데 이렇게 해야 해당 박스의 재사용이나 측정이 쉽기 때문이다. 그

런데 expl3는 \hbox{0}와 같은 방식으로 인자를 box에 바로 넣는 명령도 있기는 있다. 그것이 \hbox:n이나 \vbox:n인데 사실상 \hbox, \vbox의 별칭(alias)이다.

예제 11: \raisebox 매크로 LaTeX 명령인 \raisebox 매크로를 이용하는 예제

```
The\raisebox{.5\totalheight}[0pt]{\textdbend}Korean\par
The\raisebox{0pt}[0pt]{\textdbend}Korean\par
The\textdbend Korean\par
The\raisebox{-.5\totalheight}[0pt]{\textdbend}Korean\par
The\raisebox{-.5\totalheight}[0pt][0pt]{\textdbend}Korean\par
\leavevmode\par
```



\raisebox를 expl3로 재구현해보려 한다. 이 매크로에 대한 설명은 해당 논문에 자세하므로 충분히 예습하였을 것이다.

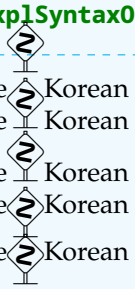
```
\ExplSyntaxOn
\NewDocumentCommand \myRaisebox { m o o +m }
{
    \hbox_set:Nn \l_tmpa_box { #4 }
    \hbox_set:Nn \l_tmpb_box
        { \box_move_up:nn { #1 } { \box_use:N \l_tmpa_box } }

    \IfValueT { #2 }
    {
        \box_set_ht:Nn \l_tmpb_box { #2 }
    }

    \IfValueT { #3 }
    {
        \box_set_dp:Nn \l_tmpb_box { #3 }
    }

    \box_use_drop:N \l_tmpb_box
}
\ExplSyntaxOff

K\myRaisebox{5pt}{O}R\myRaisebox{-5pt}{E}A
```

$K^{O}R_{E}A$

논문의 75–76에 걸쳐 설명하고 있는 \raisebox와 \@iirsbox에 대한 설명을 이해하였다면 위의 코드가 완전히 동일한 일을 하고 있는 것임을 알 수 있을 것이다. expl3와 xparse로 얼마나 이해하기 쉽고 간편한 코딩이 가능한지를 보여준다.
그런데 예제를 식자하려면 \width와 같은 매크로가 박스를 처리하기 전에 결정되어 있어야 한다. 이 문제는 다음과 같이 하여 해결할 수 있다.

```
\ExplSyntaxOn
\NewDocumentCommand \myRaisebox { m o o +m }
{
    \calc_and_set_len:n { #4 }

    \hbox_set:Nn \l_tmpa_box { #4 }
    \hbox_set:Nn \l_tmpb_box { \raise #1 \box_use:N \l_tmpa_box }

    \IfValueT { #2 }
    {
        \box_set_ht:Nn \l_tmpb_box { #2 }
    }

    \IfValueT { #3 }
    {
        \box_set_dp:Nn \l_tmpb_box { #3 }
    }


    \box_use:N \l_tmpb_box
}

\cs_new:Npn \calc_and_set_len:n #1
{
    \hbox_set:Nn \l_tmpa_box { #1 }
    \dim_zero_new:N \g_width_dim
    \dim_zero_new:N \g_height_dim
    \dim_zero_new:N \g_depth_dim
    \dim_zero_new:N \g_totalheight_dim
    \dim_set:Nn \g_width_dim { \box_wd:N \l_tmpa_box }
    \dim_set:Nn \g_height_dim { \box_ht:N \l_tmpa_box }
    \dim_set:Nn \g_depth_dim { \box_dp:N \l_tmpa_box }
    \dim_set:Nn \g_totalheight_dim { \g_height_dim + \g_depth_dim }
    \cs_gset_eq:NN \width \g_width_dim
    \cs_gset_eq:NN \height \g_height_dim
    \cs_gset_eq:NN \depth \g_depth_dim
    \cs_gset_eq:NN \totalheight \g_totalheight_dim
}
\ExplSyntaxOff
```

```
\ExplSyntaxOn
The\myRaisebox{.5\totalheight}[0pt]{\textdbend}Korean\par
The\myRaisebox{0pt}[0pt]{\textdbend}Korean\par
The\textdbend Korean\par
The\myRaisebox{-.5\totalheight}[0pt]{\textdbend}Korean\par
The\myRaisebox{-.5\totalheight}[0pt][0pt]{\textdbend}Korean\par
\leavevmode\par
\ExplSyntaxOff
```



연습문제

기본 11. 같은 논문의 예제 13, 예제 16, 예제 18을 expl3로 해보아라.

예제

프랑스어로 lettrine, 영어에서 initial이라고 부르는 조판 형태가 있다. 한 단락의 첫 글자를 특별하게 장식하는 것이다.

Lorem ipsum 텍스트를 가지고 lettrine 조판을 구현하여라. 첫 글자는 3행을 drop하여야 한다. lettrine 패키지와 같은 외부 패키지의 사용은 금지한다.

\parshape에 대한 복습 우리가 이미 잘 알고 있고 즐겨 사용하는 \parshape에 대해서 간략히 복습해 두자.

```
\parshape 5
    1cm 12cm
    2cm 8cm
    3cm 8cm
    4cm 8cm
    1cm \dimexpr\textwidth-1cm\relax
\lipsum[1]
```

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

복습이 끝났다. \parshape의 사용법을 잘 기억해낼 수 있겠는가?

첫 글자 장식 문단 전체에서 첫 글자를 가져오는 것은 그다지 어렵지 않을 것이다. 그 첫 글자를 적당한 크기(여기서는 세로로 3줄에 걸치는 정도로 하겠다)로 만들어서 box에 넣는다. 그리고 이 박스가 차지할 공간을 계산하여 parshape를 주면 될 것 같다.
먼저 문단의 첫 글자만 잘라내는 것.

```
\ExplSyntaxOn

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

\cs_set_eq:NN \captureachar \capture_a_char:n

\ExplSyntaxOff
\captureachar Lorem ipsum dolor sit amet

\ExplSyntaxOn
\tl_use:N \l_tmpa_tl
\ExplSyntaxOff
```

```
orem ipsum dolor sit amet
L
```

이제 이 글자를 충분히 큰 크기로 키워서 박스에 넣는다. 이 박스는 높이가 없어야 한다. 그런 다음에 문단의 처음 세 줄에 대하여 \parshape를 주는데 (정상문단으로 돌아오는 것까지 합쳐서 \parshape의 첫 인자는 4가 된다) 그러려면 글자 박스의 width가 필요하다. 그리고 Initial 글자는 항상 대문자가 되면 좋겠다.

```
\ExplSyntaxOn
\cs_set:Npn \capture_a_char:n #1
{
    \hbox_set:Nn \l_tmpb_box { \tl_upper_case:n { #1 } }
    \box_resize_to_ht:Nn \l_tmpb_box { 2.2\onelineskip }

    \vbox_set_top:Nn \l_tmpa_box { \vskip -3pt\box_use_drop:N \l_tmpb_box }
    \dim_set:Nn \l_tmpa_dim { \box_wd:N \l_tmpa_box + \fboxsep }
    \dim_set:Nn \l_tmpb_dim { \textwidth - \l_tmpa_dim }
    \box_set_ht:Nn \l_tmpa_box { \c_zero_dim }
    \box_set_dp:Nn \l_tmpa_box { \c_zero_dim }

    \mode_if_vertical:T { \mode_leave_vertical: }

    \parshape 4
        \l_tmpa_dim \l_tmpb_dim
        \l_tmpa_dim \l_tmpb_dim
        \l_tmpa_dim \l_tmpb_dim
        0pt \textwidth

    \hbox_overlap_left:n { \box_use:N \l_tmpa_box \hskip \fboxsep }
}

\capture_a_char:n
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.

\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 laborisnisi ut aliquip ex ea commodo consequat. Duis aute irure dolorin reprehenderit in voluptate velit esse cillum dolore eu fugiatnulla pariatur. Excepteur sint occaecat cupidatat non proident,sunt in culpa qui officia deserunt mollit anim id est laborum.

글자의 크기를 맞추기 위하여 일단 hbox에 넣고 확대한 다음 이를 vbox에 다시 넣었다. 글자를 키우는 데 `\box_resize_to_ht:Nn`을 사용하였다. scale 함수도 있으므로 interface3에서 찾아보아라. 그리고 회전(rotate)도 가능하다. 여기서는 쓰지 않았지만.

머릿글자 박스를 찍을 때 위의 예에서는 `\hbox_overlap_left:n`을 썼다. 이것은 `\llap`에 해당하는 expl3 명령이다. 머릿글자 위치의 미세조정을 위해 vbox에 넣으면서 수직 위치 조절을 위한 세로 간격을 추가한 것(이 위치는 "내부 수직 모드"이다)과 hbox에서 `\fboxsep`을 글자의 오른쪽에 둔 것을 잘 보면 된다. 기억할 것은 수직 길이는 vbox에서, 수평 길이는 hbox에서 조절하는 것이다.
이 방법은 여러 개의 문단의 머릿글자를 모두 drop하려 할 때 문단마다 initial 박스의 가로 길이가 달라질 수 있다. 판면 전체의 조화를 위하여 이 박스의 width를 고정하려 한다면 박스의 wd를 정해진 값으로 고정할 수 있다. 간단히 가능하므로 직접 시도해보기 바란다.
이제 문서 명령으로 만들어서 원하는 문단의 머리에 두면 될 것이므로 여기까지만 진행해 본다.

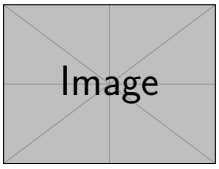
연습문제

기본 12. 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.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.



Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

읽을 거리

[1] 조진환.《LaTeX 박스 매크로 분석》. *Asian Journal of TEX*, vol. 1, no. 1, pp. 67–84. 2007. pdf online: *AJT* Archive.

[2] 김도현,《수평모드와 수직모드》. 한국텍학회 정기총회 및 학술대회 발표자료. 2017. pdf online: KTUG Wiki.

24 seq, tl, (big)int

예제

1항하사는 10^{52} 이다. 이를 다음과 같이 보이고자 한다.

1항하사 0000극 0000재 0000정 0000간 0000구 0000양 0000자 0000해 0000경 0000조 0000억 0000만 0000

임의의 큰 수를 인자로 받아들여서 위와 같이 네 자리마다 우리식 읽기 단위를 붙여주는 명령을 만들어보아라.

입력: `\mybignum{1.732051 * 10**40}`
출력: 1정 7320간 5100구 0000양 0000자 0000해 0000경 0000조 0000억 0000만 0000

24.1 big integers

자연수(양의 정수)만을 문제삼을 때 $2^{31} - 1$ (2147483647)을 넘는 수를 "큰 수"라고 한다. expl3에는 unsigned integer가 없기 때문이다.
팩토리얼 계산에서는 13!에서부터 이 범위를 넘어선다.

```
>>> import math
>>> math.factorial(12)
479001600
>>> math.factorial(13)
6227020800
```

xint는 LaTeX을 위한 상당히 강력한 수 계산 엔진이다. 여러 패키지로 구성되어 있는 이 엔진을 big integers가 필요할 때에 활용할 수 있다.
bnumexpr 패키지는 xint를 백엔드로 하여 큰 수 계산의 사용자 인터페이스를 제공한다. 이 글에서는 xint를 모두 살펴보는 것이 목표가 아니기 때문에 bnumexpr만을 활용할 것이다. preamble에

```
\usepackage{bnumexpr}
```

라고 선언하여 패키지를 활성화할 수 있다. 문서를 읽어보려면 항상 그렇듯이

```
texdoc xint
texdoc bnumexpr
```

명령을 내려서 pdf를 읽을 수 있다.
bnumexpr는 expl3로 작성되거나 expl3식 명령을 제공하지는 않는다. 다음 두 개의 명령

```
\thebnumexpr
\bnumeval
```

을 기억하는 것만으로 원하는 계산을 거의 대부분 할 수 있다. 또한 다음 연산자를 제공하여 편의를 더하고 있다.

- 지수 계산 연산자: ** 또는 ^

- 팩토리얼 연산자: !
- trunc 나눗셈과 modulo 연산자: //, /:

큰 수를 입력하다가보면 자릿수를 헷갈릴 가능성이 매우 크다. 언더브레이스를 이용하여 입력시의 시각적 구분을 가능하게 한다.

```
\bnumeval{1_000_000_000_000_000_000_000}
```

```
1000000000000000000
```

아주 긴 수를 행을 나누어 인쇄하는 방법에 대해서는 패키지 설명서에 방법이 소개되어 있다.

이제, 1항하사를 숫자로 표현하여 이것을 tl에 넣어 보관하기로 하자.

```
\ExplSyntaxOn
\tl_set:Nf \l_longnum_tl { \bnumeval{ 10**52 } }
%%% test
\tl_use:N \l_longnum_tl {}~( \tl_count:N \l_longnum_tl )
\ExplSyntaxOff
```

```
10000000000000000000000000000000000000000000000000000 (53)
```

\bnumeval은 "두 번" 확장이 필요하기 때문에 o 확장으로는 안 된다. 그래서 f 확장을 적용하였다.

큰 수의 단위 초등학생에게나 신기할 지식이기는 하지만, 큰 수는 10^4 마다 다음 단위를 차례로 붙여가는 것으로 알려져 있다.

> 만(萬), 억(億), 조(兆), 경(京), 해(垓), 자(秭), 양(穰), 구(溝), 간(澗), 정(正), 재(載), 극(極), 항하사(恒河沙), 아승기(阿僧祇), 나유타(那由他), 불가사의(不可思議), 무량(대)수(無量數)

'만' 에서부터 '재' 까지는《손자산경》에 근거가 있으며 "극"에서 "무량수"까지 수사가 처음 나타나는 것은 (아마도) 원대의《산학계몽》이 아닌가 싶다. 그러나 산학계몽의 수를 세는 방법은 이른바 "억진법"이라고 하여 "억"까지는 그대로 세지만 그 이후로는 "억 단위로 반복" 하는 것이라 한다.

> 억, 십억, 백억, 천억, 만억, 십만억, 백만억, 천만억, 조

현재 우리나라에서 천억 다음 단위가 "만억"이 아니라 "조"이기 때문에 이미 이 방법은 쓰고 있지 않은 것이며 간략하게 10^4 마다 새로운 단위를 붙여가는 "만진법"을 쓰고 있는 것으로 간주해도 좋을텐데, 이 방법이 확립된 것은 일본의 산서《진겁기》로 보아야 할 것 같다. 현재 우리나라에서 쓰이는 수를 세는 방법은 이 일본의 방법이 개화기에 들어온 것이 유지되고 있는 것으로 본다.
우리말에 "억만장자"라는 말이 있는데 억진법의 "만억"을 잘못 쓴 것으로 이해한다면 "만억"은 현재의 "조"에 해당하는 값이다.
아무튼 항하사라는 단위에 대해서만 조금 더 설명하고자 한다.
$1cm^3$에 들어 있는 모래알이 대략 5,000개라고 하자. 갠지스강의 길이가 2,500km이고 강변에 평균적으로 모래가 깊이 2m, 너비 5m 정도로 고르게 분포한다고 가정하면, 모래알의 개수는 50000000000000 개 정도일 것이다. 그러니 "항하사(gaganānadīvālukopama)"라는 단위 이름은 허황된 것이 틀림없다. 이 표현 자체는 대반야경과 금강경 등 여러 경전에 나온다. 다음은 금강경의 구절이다.

> 須菩提 如恒河中所有沙數 如是沙等恒河 於意云何. 是諸恒河沙 寧爲多不. 須菩提言 甚多世尊 但諸恒河 尙多無數 何況其沙. (11장 無爲福勝分)

거칠게 해석하면,

수보리여, 갠지스 강 속의 모래의 수와 같이 그만큼 수의 갠지스 강이 있다면, 네 의견에는 어떠냐? 이 모든 갠지스 강의 모래들이 많다 하겠느냐 그렇지 않으냐?

수보리 말하기를, 심히 많으옵니다, 세존이시여, 강들의 수만으로도 무수히 많겠거늘 하물며 그 모래알의 수이겠사옵니까?

밑줄 친 단어가 "항하사"이다.

24.2 문자열을 잘라내어 seq에 넣기

보조문제

abcdefghijklmnopq와 같이 주어지는 문자열이 있다. 이 문자열을 n개씩 잘라내어 지정되는 seq에 넣는 명령을 작성하여라.

입력: `\split_tl_to_seq:nnN { abcdefghijklmnopq } { 3 } \l_output_seq`
출력: `\seq_use:Nn \l_output_seq {,~ }` → abc,def,ghi,jkl,mno,pq

이런 비슷한 문제를 이미 풀어보았다. 여러 가지 방법을 조금 더 생각해보려고 한다.

tl map 해결 이미 배운 방법으로서 tl을 하나씩 읽어가면서 n번째가 되면 그 때까지 쌓인 tl을 모아서 seq에 put right하는 방법이다. 설명이 필요없을 것으로 본다.

```
\ExplSyntaxOn

\seq_clear:N \l_tmpa_seq

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

\tl_map_inline:nn { abcdefghijklmnopq }
{
    \int_incr:N \l_tmpa_int
    \int_compare:nTF { \l_tmpa_int == 3 }
    {
        \tl_put_right:Nn \l_tmpa_tl { #1 }
        \seq_put_right:Nx \l_tmpa_seq { \l_tmpa_tl }
        \tl_clear:N \l_tmpa_tl
        \int_zero:N \l_tmpa_int
    }
    {
        \tl_put_right:Nn \l_tmpa_tl { #1 }
    }
}

\tl_if_empty:NF \l_tmpa_tl
{
    \seq_put_right:Nx \l_tmpa_seq { \l_tmpa_tl }
}

\seq_use:Nn \l_tmpa_seq { |~ }

\ExplSyntaxOff
```

abc| def| ghi| jkl| mno| pq

위의 코드를 함수(cs) 형태로 묶는 것은 우리가 충분히 연습했고 어려운 일이 아니므로 더 진행하지 않겠다. 이 방법은 맨처음 배운 것이고 이해하기 가장 쉽지만 뭔가 여러 가지 변수를 clear하고 zero하고 하는 번거로운 과정이 포함되어 있어서 아주 세련되어 보이지는 않는다.

regex 활용 위의 방법보다 훨씬 세련되어 보이는 regex 방법을 시도해보자.

```
\ExplSyntaxOn
\seq_clear:N \l_tmpa_seq

\cs_new:Npn \get_three_to_seq:w #1 \q_stop
{
    \seq_put_right:Nn \l_tmpa_seq { #1 }
}

\tl_set:Nn \l_tmpa_tl { abcdefghijklmnopq }
\int_set:Nn \l_tmpa_int { \tl_count:N \l_tmpa_tl }

\regex_replace_all:nnN
    { (.)(.)(.) }
    { \c{get_three_to_seq:w} \0 \c{q_stop} }
    \l_tmpa_tl

\int_compare:nF { \int_mod:nn { \l_tmpa_int } { 3 } == 0 }
{
    \regex_replace_once:nnN
        { (.*) \c{q_stop} (.+?) $ }
        { \1 \c{q_stop} \c{get_three_to_seq:w} \2 \c{q_stop} }
        \l_tmpa_tl
}

\tl_use:N \l_tmpa_tl >>
\seq_use:Nn \l_tmpa_seq {|~}
\ExplSyntaxOff
```

»abc| def| ghi| jkl| mno| pq

이 해법이 의도하는 것은 다음과 같다. 먼저 `\get_three_to_seq:w #1\q_stop`이라는 형태의 함수를 정의하여 둔다. 이게 하는 일은 `\q_stop`까지의 토큰을 seq에 넣는 것이다.
그리고 예를 들어 abcdefghijklmkopq라는 토큰열이 있다면 이것을

```
\get_three_to_seq:w abc\q_stop
\get_three_to_seq:w def\q_stop
\get_three_to_seq:w ghi\q_stop
\get_three_to_seq:w jkl\q_stop
\get_three_to_seq:w mno\q_stop
\get_three_to_seq:w pq\q_stop
```

라는 형태로 만들려고 하는 것이다.
먼저 세 개씩 묶어서 그 앞뒤에 원하는 매크로를 붙여놓고 한 번 확장해보면

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { abcdefghijklmnopq }
\regex_replace_all:nnN
    { (.)(.)(.) }
    { \c{get_three_to_seq:w} \0 \c{q_stop} }
    \l_tmpa_tl
```

```
{\ttfamily
    \exp_args:No \tl_to_str:n { \l_tmpa_tl }
}
\ExplSyntaxOff
```

```
\get_three_to_seq:w abc\q_stop \get_three_to_seq:w def\q_stop
\get_three_to_seq:w ghi\q_stop \get_three_to_seq:w jkl\q_stop
\get_three_to_seq:w mno\q_stop pq
```

이 케이스에서 마지막에 pq에는 원하는 매크로가 붙어 들어 있지 않다. 당연한 것이 "3개"를 취하라는 regex 명령이 완수될 수 없었기 때문이다. 만약 토큰열이 3의 배수개였다면 걱정할 것이 없다.

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { abcdefghijklmnopqr }
\regex_replace_all:nnN
    { (.)(.)(.) }
    { \c{get_three_to_seq:w} \0 \c{q_stop} }
    \l_tmpa_tl

{\ttfamily
    \exp_args:No \tl_to_str:n { \l_tmpa_tl }
}
\ExplSyntaxOff
```

```
\get_three_to_seq:w abc\q_stop \get_three_to_seq:w def\q_stop
\get_three_to_seq:w ghi\q_stop \get_three_to_seq:w jkl\q_stop
\get_three_to_seq:w mno\q_stop \get_three_to_seq:w pqr\q_stop
```

따라서 토큰열의 count가 3의 배수가 아닐 때에만 마지막 남은 것들을 처리하는 조치가 필요하다. 그것이 `\int_compare:nF` 문장이다. 제대로 되는지 보겠다.

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { abcdefghijklmnopq }
\int_set:Nn \l_tmpa_int { \tl_count:N \l_tmpa_tl }

\regex_replace_all:nnN
    { (.)(.)(.) }
    { \c{get_three_to_seq:w} \0 \c{q_stop} }
    \l_tmpa_tl

\int_compare:nT { \int_mod:nn { \l_tmpa_int } { 3 } != 0 }
{
    \regex_replace_once:nnN
        { (.*) \c{q_stop} (.+?) $ }
        { \1 \c{q_stop} \c{get_three_to_seq:w} \2 \c{q_stop} }
        \l_tmpa_tl
}

{\ttfamily
    \exp_args:No \tl_to_str:n { \l_tmpa_tl }
}
\ExplSyntaxOff
```

```
\get_three_to_seq:w abc\q_stop \get_three_to_seq:w def\q_stop
\get_three_to_seq:w ghi\q_stop \get_three_to_seq:w jkl\q_stop
\get_three_to_seq:w mno\q_stop \get_three_to_seq:w pq\q_stop
```

두 번째 `\regex_replace_once:nnN` 함수를 원래 추어진 문자열의 (다시 말하면 regex에 의해서 수정되기 전의) 길이가 3의 배수가 아닐 때만 실행되게 하면 된다.
참고로, `\tl_to_str:n` 함수는 토큰을 "있는 그대로" 식자해준다. `\exp_args:No`를 앞에 두었기 때문에 `\l_tmpa_tl`이 한 번 확장된 단계에서의 토큰열을 매크로로서 실행하지 않고 출력해주기 때문에 디버깅 때 이따금 유용하게 쓸 수 있다.
이렇게 수정된 tl을 `\tl_use:N`으로 확장하면 `\get_three_to_seq:w` 함수가 확장되면서 seq안에 원하는 대로 아이템들이 들어가 있게 된다. 매우 재미난 방법이다.

`\tl_range:nnn` 함수를 이용하자 이번에는 `\tl_range:nnn`이라는 함수를 이용해보기로 한다. 이 함수는 우리가 이미 만들어 본 적이 있는 문자열 slicing 함수인데 interface function으로 제공되는 것이 이미 있다.
로직은 간단하다. 3글자씩 자른다면 start를 1, last를 3으로 처음에 설정하고 그 후로 4, 6; 7, 9; 같은 식으로 3씩 더해가면서 slicing해서 이를 출력하자는 것이다.
seq에 넣는 것을 잠시 미루고 일단 그대로 입력 스트림에 남기는 것만 테스트해본다.

```
\ExplSyntaxOn
\int_new:N \l_start_int
\int_new:N \l_last_int

\cs_new:Npn \get_three:n #1
{
    \int_set:Nn \l_start_int { 1 }
    \int_set:Nn \l_last_int { 3 }

    \int_step_inline:nn { \tl_count:n { #1 } }
    {
        \int_compare:nTF { \l_last_int > \tl_count:n { #1 } }
        {
            \tl_range:nnn { #1 } { \l_start_int } { \l_last_int }
            \esg_int_step_break:
        }
        {
            \tl_range:nnn { #1 } { \l_start_int } { \l_last_int } |
            \int_add:Nn \l_start_int { 3 }
            \int_add:Nn \l_last_int { 3 }
        }
    }
}

\get_three:n { abcdefghijklmnop }
\ExplSyntaxOff
```

```
abc|def|ghi|jkl|mno|p
```

여기서는 `\int_step_inline:nn`이 쓰였으니 여기서 중요한 것은 반복 횟수이다. 몇 번이나 반복하면 될까? 논리적으로 말하자면 최대 (tl길이/n + 1)번만 반복하면 된다. 그런데 이 코드에서는 어차피 중단이 `\esg_int_stop_break:`로 일어나기 때문에 (tl길이)만큼 반복시켜놓고 나중에 중단하는 방식으로 처리하였다.

두 개의 seq로부터 새로운 seq 구성 처음 문제를 해결해보자. 지금 우리는 두 개의 seq를 구성할 수 있다. 하나는 단위 수사를 가지고 있고 다른 하나는 수로 표현된 것을 역순으로 네 개씩 잘라서 가지고 있는 것이라고 하자.

```
\ExplSyntaxOn
\seq_use:Nn \l_numunit_seq {,~ }
\par
\seq_use:Nn \l_splittednum_seq {,~}
\ExplSyntaxOff
```

, 만, 억, 조, 경, 해, 자, 양, 구, 간, 정, 재, 극, 항하사, 아승기, 나유타, 불가사의, 무량대수
0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 1

어떻게든 이렇게 두 개의 seq가 조성되어 있다면 다음과 같이 하여 우리가 원하는 결과를 얻을 수 있다.

```
\ExplSyntaxOn
\cs_new:Npn \build_result:nn #1 #2
{
    {\tiny #2 } #1 \c_space_token
}

\seq_mapthread_function:NNN \l_splittednum_seq \l_numunit_seq
 ↪  \build_result:nn

\ExplSyntaxOff
```

0000 만0000 억0000 조0000 경0000 해0000 자0000 양0000 구0000 간0000 정0000 재0000 극0000 항하사1

이것을 뒤집으면 된다. 뒤집는 것은 아래에서 보이고 여기서는 `\seq_mapthread_function:NNN` 이라는 재미있는 함수를 눈여겨보아두자. 두 개의 seq로부터 순서대로 하나씩 취하여 (두 개의 인자를 처리하는) 어떤 function에 먹이는 것이다.

```
\ExplSyntaxOn

\seq_clear:N \l_tmpa_seq

\cs_set:Npn \build_result:nn #1 #2
{
    \seq_put_right:Nn \l_tmpa_seq { #1 { \tiny #2 } \c_space_token }
}

\seq_mapthread_function:NNN \l_splittednum_seq \l_numunit_seq
 ↪  \build_result:nn

\seq_reverse:N \l_tmpa_seq

\seq_use:Nn \l_tmpa_seq { }
\ExplSyntaxOff
```

1항하사 0000극 0000재 0000정 0000간 0000구 0000양 0000자 0000해 0000경 0000조 0000억 0000만 0000

이 결과를 보고 왜 `\l_numunit_seq`의 첫 항목이 빈 항목 (`\empty`)이었는지 이해할 수 있겠는가?

24.3 종합

변수와 사용할 함수 등을 선언한다.

```
\int_new:N \l_start_int
\int_new:N \l_last_int
```

```
\seq_new:N \l_result_seq

\seq_const_from_clist:Nn \c_bnumunit_seq
{  \empty, 만, 억, 조, 경, 해, 자, 양, 구, 간, 정, 재, 극, 항하사, 아승기, 나유타, 불
 ↪  가사의, 무량대수 }

\cs_set:Npn \build_result:nn #1 #2
{
    \seq_put_right:Nn \l_result_seq { #1 \ensuremath{ \text{
     ↪  \tiny\color{blue!80} #2 } } }
}
```

\build_result:nn 함수에서 #2에 대하여 가한 조작에서 유념할 부분은 이 결과가 "수식" 모드에서 식자될 수 있다는 가능성에 대응하여야 한다는 것이다. 그래서 한글이 오게 되는 부분에 대하여 \ensuremath와 \text를 중첩해서 썼다.
명령을 정의하고 인자를 받아들여 큰 수를 "수"의 형식으로 쓴 다음 tl에 넣고 한 번 뒤집는다.

```
\NewDocumentCommand \mybignum { m }
{
    \tl_set:Nf \l_thebignum_tl { \bnumeval { #1 } }

    \tl_reverse:N \l_thebignum_tl
```

변수 초기화

```
\int_set:Nn \l_start_int { 1 }
\int_set:Nn \l_last_int { 4 }

\seq_clear:N \l_tmpa_seq
\seq_clear:N \l_result_seq
```

네 개씩 잘라서 \l_tmpa_seq에 넣는다.

```
\int_step_inline:nn { \tl_count:N \l_thebignum_tl }
{
    \int_compare:nTF { \l_last_int > \tl_count:N \l_thebignum_tl }
    {
        \seq_put_right:Nx \l_tmpa_seq
        {
            \exp_args:No
            \tl_range:nnn { \l_thebignum_tl } { \l_start_int } { \l_last_int
             ↪  }
        }
        \esg_int_step_break:
    }
    {
        \seq_put_right:Nx \l_tmpa_seq
        {
            \exp_args:No
            \tl_range:nnn { \l_thebignum_tl } { \l_start_int } { \l_last_int
             ↪  }
        }
        \int_add:Nn \l_start_int { 4 }
        \int_add:Nn \l_last_int { 4 }
    }
}
```

네 개씩 잘린 seq와 단위를 저장하고 있는 seq 두 개로부터 결과가 될 seq를 구성한 다음 결과를 한 번 뒤집는다.

```
\seq_mapthread_function:NNN \l_tmpa_seq \c_bnumunit_seq \build_result:nn

\seq_reverse:N \l_result_seq
```

다 되었다. 이제 출력하고 명령의 정의를 마친다.

```
    \seq_use:Nn \l_result_seq {\,}
}
```

이 명령으로써 문서 중에 "1항하사"를 숫자로 보여주면서 큰 수 읽는 법도 함께 표현할 수 있다.

```
\[ \mybignum{10**52} \]
```

1항하사 0000극 0000재 0000정 0000간 0000구 0000양 0000자 0000해 0000경 0000조 0000억 0000만 0000

색상이나 단위의 장식 등은 적당한 곳에서 원하는 대로 수정할 수 있을 것이다.
이 예제에서 보이고자 하는 것은 seq라는 자료형이 얼마나 유용하고 중요한 것인가 하는 점이다. expl3를 씀으로써 얻을 수 있는 거의 대부분의 장점은 seq (또는 clist)를 이용할 수 있다는 점이라 해도 과언이 아니다. interface3 문서에서 seq 관련 함수들을 찬찬히 복습해두는 것이 좋다.

24.4 보너스: tl로만 해보자

그런데 seq보다 tl이 더 좋은 사람도 얼마든지 있을 것이다. 오로지 tl만으로 주어진 문제를 해결할 수는 없을까? 단 큰 수를 읽는 "단위"는 clist에 들어 있다고 가정한다.
다음과 같은 방법이 가능하였다.

```
\ExplSyntaxOn
\tl_set:Nx \l_tmpa_tl { \thebnumexpr 10**52\relax }
\tl_reverse:N \l_tmpa_tl
\int_zero:N \l_tmpa_int
\tl_clear:N \l_tmpb_tl

\clist_set:Nn \l_tmpa_clist { 만, 억, 조, 경, 해, 자, 양, 구, 간, 정, 재, 극, 항하
 ↪ 사, 아승기, 나유타, 불가사의, 무량수 }
\tl_new:N \l_tmpc_tl

\tl_map_inline:Nn \l_tmpa_tl
{
    \int_incr:N \l_tmpa_int
    \int_compare:nTF { \l_tmpa_int < 4 }
    {
        \tl_put_right:Nn \l_tmpb_tl { #1 }
    }
    {
        \tl_put_right:Nn \l_tmpb_tl { #1\c_space_token }
        \clist_pop:NN \l_tmpa_clist \l_tmpc_tl
        \tl_put_right:Nx \l_tmpb_tl { { \exp_not:N \tiny \l_tmpc_tl } }
        \int_zero:N \l_tmpa_int
    }
}
\tl_reverse:N \l_tmpb_tl
```

```
\tl_use:N \l_tmpb_tl
\ExplSyntaxOff
```

1항하사 0000극 0000재 0000정 0000간 0000구 0000양 0000자 0000해 0000경 0000조 0000억 0000만 0000

이 기법에서 나름 어려운(?) 것은 clist에서 뽑아낸 단위 이름이 tl로 풀렸을 때 이를 reverse하면 함께 뒤집혀서 "1사하항"으로 나오는 참사를 막아야 한다는 것과, `\tiny \l_tmpc_tl`과 같이 tl의 항목으로 넣었을 때 `\tiny`보다 `\l_tmpc_tl`이 미리 풀려야 한다는 문제를 해결해야 하는 것이다. "사하항"으로 뒤집히지 않게 하기 위해서 `\l_tmpc_tl`을 중괄호로 묶어주었다. 또 `\tiny`의 확장 문제는 `\tl_put_right:`의 인자 확장을 `:Nx`로 하되 `\tiny`에는 `\exp_not:N`을 주었다. 이 부분은 다음과 같이 하는 방법도 있다.

```
\tl_put_right:No \l_tmpb_tl
   { \exp_after:wN { \exp_after:wN \tiny \l_tmpc_tl } }
```

짧아서 간단해보여도 seq를 이용하는 것보다 쉽다고 하기가 어렵다.

25 Permutation

예제

네 개의 문자 'a' , 'b' , 'c' , 'd' 로 만들 수 있는 모든 문자열을 출력하여라.

25.1 for 문의 중첩

움베르토 에코의 소설《푸코의 진자》의 주인공 벨보가 비밀번호를 알아내려고 당시의 컴퓨터(아마도 8bit 애플 II 정도의 기계인 듯)를 이용하여 basic 프로그램을 짜는 부분이 있다. 그 basic 코드를 python으로 다시 써보면 다음과 같다.

```
>>> a=['a','b','c','d']
>>> for i in range(1,5):
        for j in range(1,5):
                if i !=j :
                        for k in range(1,5):
                                if k!=j and k!=i:
                                        l = 10 - (i+j+k)
                                        a[i-1],a[j-1],a[k-1],a[l-1]
```

훌륭한 해법이라서가 아니라 `\int_step_...`의 중첩을 연습하는 의미에서 expl3로 써본다.

```
\ExplSyntaxOn

\clist_set:Nn \l_tmpa_clist { a, b, c, d }

\int_new:N \l_i_int
\int_new:N \l_j_int
\int_new:N \l_k_int
\int_new:N \l_l_int

\cs_set:Npn \sub_fn_iii:n #1
{
    \int_gset:Nn \l_k_int { #1 }
    \bool_if:nT
    {
        \int_compare_p:n { \l_i_int != \l_k_int }
        &&
        \int_compare_p:n { \l_j_int != \l_k_int }
    }
    {
        \int_set:Nn \l_l_int { 10 - ( \l_i_int + \l_j_int + \l_k_int ) }
        \clist_item:Nn \l_tmpa_clist { \l_i_int },
        \clist_item:Nn \l_tmpa_clist { \l_j_int },
        \clist_item:Nn \l_tmpa_clist { \l_k_int },
        \clist_item:Nn \l_tmpa_clist { \l_l_int } \quad
    }
}

\cs_set:Npn \sub_fn_ii:n #1
{
    \int_gset:Nn \l_j_int { #1 }
    \int_compare:nT { \l_i_int != \l_j_int }
    {
        \int_step_function:nN { 4 } \sub_fn_iii:n
    }
```

```
}

\cs_set:Npn \sub_fn_i:n #1
{
    \int_gset:Nn \l_i_int { #1 }
    \int_step_function:nN { 4 } \sub_fn_ii:n
}

\int_step_function:nN { 4 } \sub_fn_i:n
\ExplSyntaxOff
```

```
a,b,c,d  a,b,d,c  a,c,b,d  a,c,d,b  a,d,b,c  a,d,c,b  b,a,c,d  b,a,d,c  b,c,a,d  b,c,d,a  b,d,a,c
b,d,c,a  c,a,b,d  c,a,d,b  c,b,a,d  c,b,d,a  c,d,a,b  c,d,b,a  d,a,b,c  d,a,c,b  d,b,a,c  d,b,c,a
d,c,a,b  d,c,b,a
```

expl3의 반복 함수에서 인자를 ##1까지밖에 쓸 수 없다는 제한 때문에 `\int_step_inline:nn`을 두 번 이상 중첩해서 쓸 수는 없다. `\int_step_function:nN`으로 어떻게 했는지를 위에서 볼 수 있다.

25.2 재귀

다음 코드는 GeekofGeeks 사이트에서 찾은 permutation 소스코드이다. (python3를 위하여 조금 수정하였다.)

```
# Python program to print all permutations with
# duplicates allowed
def toString(List):
    return ''.join(List)

# Function to print permutations of string
# This function takes three parameters:
# 1. String
# 2. Starting index of the string
# 3. Ending index of the string.
def permute(a, l, r):
    if l==r:
        print(toString(a))
    else:
        for i in range(l,r+1):
            a[l], a[i] = a[i], a[l]
            permute(a, l+1, r)
            a[l], a[i] = a[i], a[l] # backtrack

# Driver program to test the above function
string = "ABCD"
n = len(string)
a = list(string)
permute(a, 0, n-1)

# This code is contributed by Bhavya Jain
```

이것을 expl3로 옮기는 것은 연습문제로 한다. 참고로 KTUG Q&A:233406에 이미 해결되어 있기는 하지만 되도록 이를 보지 말고 직접 해보기 바란다.

위의 python 코드에서 `a[l], a[i] = a[i], a[l]` 부분이 있는데 이것은 expl3의 표현으로 하자면 a라는 tl의 i번째 항목과 j번째 항목을 바꾸라는 것이다.

```
\cs_new:Npn \tlswap_fn:nnnN #1 #2 #3 #4
{
    \tl_clear:N #4
    \tl_set:Nx \l_tmpa_tl { #3 }

    \int_step_inline:nn { \tl_count:N \l_tmpa_tl }
    {
        \int_case:nnF { ##1 }
        {
            { #1 } { \tl_put_right:Nx #4 { \tl_item:Nn \l_tmpa_tl { #2 } } }
            { #2 } { \tl_put_right:Nx #4 { \tl_item:Nn \l_tmpa_tl { #1 } } }
        }
        {
            \tl_put_right:Nx #4 { \tl_item:Nn \l_tmpa_tl { ##1 } }
        }
    }
}
```

위의 보조함수가 하는 일은

```
\tlswap_fn:nnnN { 2 } { 4 } { abcdef } \l_result_tl
```

여기서 "abcdef"의 2번째 항목 b와 4번째 항목 d를 교환한 adcbef라는 문자열을 만들어서 `\l_result_tl`에 넣어 반환하는 것이다.
이것으로써 항목의 교환(swap) 문제는 해결할 수 있으므로 주어진 문제를 푸는 데 어려움이 없을 것으로 생각한다.
또한 주어진 python 코드에서 backtrack이라고 하여 원래로 되돌리는 코드를 expl3에서는 피해갈 수 있다. 그 이유는 위의 swap 함수를 잘 보면 swapped된 문자열이 원래의 문자열을 변경하는 것이 아니기 때문이다. 새로운 tl변수에 결과를 넣어 처리하고 있다.
원한다면 원래의 문자열 자체를 변경하도록 작성할 수 있으나 그렇게 하면 backtrack을 꼭 해야 한다.
마지막으로 tl이 아니라 seq를 이용하고 싶다면 그렇게 하여도 좋다.

25.3 Heap Algorithm

위에 소개한 KTUG의 대화에 남수진 선생이 lua로 heap algorithm을 소개하고 있다. 이 알고리즘을 python으로 쓰면 대략 다음과 같이 되는데

```
# Heap's algorithm

#Prints the array
def printArr(a, n):
    for i in range(n):
        print(a[i],end=" ")
    print()

# Generating permutation using Heap Algorithm
def heapPermutation(a, size, n):
    # if size becomes 1 then prints the obtained
    # permutation
    if (size == 1):
```

```
        printArr(a, n)
        return

    for i in range(size):
        heapPermutation(a,size-1,n);

        # if size is odd, swap first and last
        # element
        # else If size is even, swap ith and last element
        if size&1:
            a[0], a[size-1] = a[size-1],a[0]
        else:
            a[i], a[size-1] = a[size-1],a[i]


# Driver code
a = [1, 2, 3, 4]
n = len(a)
heapPermutation(a, n, n)
```

이것을 expl3로 옮겨 보았더니 다음과 같았다.

```
\ExplSyntaxOn

%% Heap's algorithm for generating permutations

\NewDocumentCommand \heapperm { m o }
{
    \tl_set:Nn \l_tmpa_tl { #1 }

    %% \l_tmpc_int counts generated permutations. therefore
    %% the last counter should indicate n!.
    \int_zero:N \l_tmpc_int

    %% if #2 exists then generates permutations of the first #2 elements
    %% adjoining the last elements to each of them.
    \IfValueTF { #2 }
    {
        \h_permute_fn:Nn \l_tmpa_tl { #2 }
    }
    {
        \h_permute_fn:No \l_tmpa_tl { \tl_count:N \l_tmpa_tl }
    }
}

%% #1: N-type tl
%% #2: size (int)
\cs_new:Npn \h_permute_fn:Nn #1 #2
{
    \int_compare:nTF { #2 <= 1 }
    {
        %% when size(#2) reaches 1, prints the obtained permutation
        \print_result:N #1
    }
    {
```

```
        \int_step_inline:nn { #2 }
        {
            %% recursive call here
            \h_permute_fn:No #1 { \int_eval:n { #2 - 1 } }

            %% if size(#2) is odd, swap first and last element
            \int_if_odd:nTF { #2 }
            {
                \h_tl_swappos:Nnn #1 { 1 } { #2 }
            }
            %% else, swap i-th and last element
            {
                \h_tl_swappos:Nnn #1 { ##1 } { #2 }
            }
       }
    }
}

%% aux vars
\tl_new:N \l__tmp_tl
\int_new:N \l_tmpc_int

%% #1: N-type tl
%% #2 & #3: indexes (int)
%% swap the #2-th and #3-th element in tl #1
\cs_new_nopar:Npn \h_tl_swappos:Nnn #1 #2 #3
{
    \tl_clear:N \l__tmp_tl
    \int_step_inline:nn { \tl_count:N #1 }
    {
        \int_case:nnF { ##1 }
        {
            { #2 }  { \tl_put_right:Nx \l__tmp_tl { \tl_item:Nn #1 { #3 } } }
            { #3 }  { \tl_put_right:Nx \l__tmp_tl { \tl_item:Nn #1 { #2 } } }
        }
        {
            \tl_put_right:Nx \l__tmp_tl { \tl_item:Nn #1 { ##1 } }
        }
    }
    \tl_set_eq:NN #1 \l__tmp_tl
}

\cs_generate_variant:Nn \h_permute_fn:Nn { No }

\cs_new:Npn \print_result:N #1
{
    \int_zero:N \l_tmpb_int
    \int_incr:N \l_tmpc_int

    \tl_map_inline:Nn #1
    {
        \int_incr:N \l_tmpb_int
        ##1
        \int_compare:nTF { \l_tmpb_int == \tl_count:N #1 }
        { \quad (\int_use:N \l_tmpc_int)\par }
        { ,~ }
    }
}

\ExplSyntaxOff
```

```
\heapperm{123}
```

1, 2, 3 (1)
2, 1, 3 (2)
3, 1, 2 (3)
1, 3, 2 (4)
2, 3, 1 (5)
3, 2, 1 (6)

코드가 길어진 이유 중 하나가 주어진 list의 i번째 아이템과 j번째 아이템을 교환(swap)하는 함수를 만들어서 써야 했기 때문이다. python은 한 줄이면 되는 거라서 길어 보일 뿐이다.
알고리즘을 잘 이해하고 있다면 코드 자체는 어려운 데가 없다.

연습문제

25.2 소절에서 소개하는 알고리즘으로 permutation 함수를 만들어보아라. (되도록 seq를 이용하여 해결하는 것을 권장한다.)

26 기생충

연습문제

영와 〈기생충〉에 근세와 기택이 지하실에서 모스 부호를 이용하여 편지를 주고받는 장면이 나온다. 모스 부호의 dot (dit)을 1, dash (dah)를 2, gap을 0으로 부호화하기로 하자. intercharacter gap은 표현하지 않기로 하고 interletter gap을 0, interword gap은 00으로 표현하기로 하면 0, 1, 2 세 가지 부호로 모든 모스 부호를 코드화할 수 있다.
아래 모스 부호표를 참고로 하여 `\encodeMorse` 명령과 `\decodeMorse` 명령을 작성하여라. 사용 언어는 영어와 숫자로 한다.

표 1: Morse Code Table

A	12	J	1222	S	111	1	12222
B	2111	K	212	T	2	2	11222
C	2121	L	1211	U	112	3	11122
D	211	M	22	V	1112	4	11112
E	1	N	21	W	122	5	11111
F	1121	O	222	X	2112	6	21111
G	221	P	1221	Y	2122	7	22111
H	1111	Q	2212	Z	2211	8	22211
I	11	R	121	0	22222	9	22221

원래 이 문제는 한글 모스 부호로 번역하거나 이를 복원하는 명령을 만들면서 유니코드 한글 문자를 expl3로 취급하는 문제를 다루려고 의도하였던 것이다.
사정상 위의 내용이 (초급)과정에서 제외되었으므로 이에 대한 본격적인 논의는 “중급과정”에서 기회가 닿으면 해보기로 하고 여기서는 원래 작성했던 문제의 아이디어만 소개하고 지나가는데 영문자만이 문제가 되면 특별히 어려울 것이 없어서 연습문제로 남긴다.

> GOOD MORNING
> 221022202220211002202220121021011021022 1

dot을 [U+2022] (Bullet), dash를 [U+2014] (EM-dash)로 나타내기로 하면 위의 코드는 다음과 같이 표현된다.
——•␣———␣———␣—•• ␣␣——␣———␣•—• ␣—•␣ ••␣—•␣——•

27 l3draw

예제

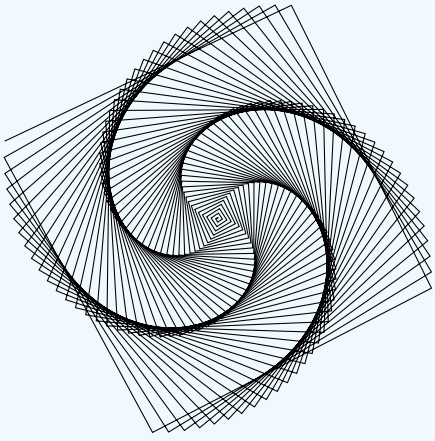
앞서 우리가 esg004에서 그려보았던 Ti*k*Z 그림을 l3draw를 이용하여 구현하여 보아라.

우리가 그림 그리기를 위하여 Ti*k*Z를 채택한 이유는 현대의 LaTeX 사용 상황에서 Ti*k*Z가 *sine qua non*이라고 보기 때문이고 이를 숙련되게 익혀두는 것이 당연히 요구되는 바이기 때문에 그랬다.
그런데 이미 한 번 해보아서 잘 알겠지만 Ti*k*Z의 바탕이 되어 있는 pgf 엔진과 expl3가 서로 달라서 이따금 충돌이 발생하기도 하기 때문에 매우 조심스럽게 다루지 않으면 안 되었다. expl3와 LaTeX 3를 만들고 있는 the LaTeX 3 Project Team에서는 Ti*k*Z에 의존하지 않고 드로잉을 할 수 있도록 하는 expl3 그리기 언어를 만들고 있는 중이다. 이를 l3draw라고 한다. 현재 간단한 직선, 곡선, 다각형 정도를 그리는 정도의 용도에 쓸 수 있을 정도로서 Ti*k*Z와 비교할 수는 없는 것이 사실. l3draw는 아직 안정화되어 있지 않아서 experimental 군에 분류되어 있으므로 이를 활성화하려면

```
\usepackage{l3draw}
```

가 필요하다. 도움문서도 interface3가 아닌 l3draw 문서가 별도로 있으므로 이를 참고한다. experimental인 experimental이라니.....
생각건대 expl3가 Ti*k*Z 수준의 그림 언어를 제공해야 할 이유는 없을 것이다. 왜냐하면 어차피 expl3라는 것이 최종사용자가 문서를 작성하면서 입력해넣을 명령을 제공하는 목적이 아니기 때문이다. 이러한 용도상의 특징을 이해하고 이를 필요한 곳에 활용할 수 있도록 하자.

```
\ExplSyntaxOn
\draw_begin:
    \draw_path_moveto:n { 0,0 }

    \dim_zero:N \l_tmpa_dim % length
    \int_zero:N \l_tmpa_int % angle (degree)
    \color_fill:n { cyan!25!magenta }
    \int_step_inline:nn { 200 }
    {
        \int_set:Nn \l_tmpa_int { #1 * 89 }
        \dim_add:Nn \l_tmpa_dim { 0.15mm }
        \draw_path_lineto:n
            { \draw_point_polar:nn { \l_tmpa_dim } { \l_tmpa_int } }
    }
    \draw_path_use_clear:n { stroke }
\draw_end:
\ExplSyntaxOff
```



여기서 이동할 목표점은 극좌표로 주어진다. 맨 첫 줄의 `0,0`은 데카르트 좌표이고
`\draw_point_polar:nn { dim } { angle }`은 극좌표이다.[3]
l3draw에서 특히 주의할 점은 Ti*k*Z와 달리 기본 길이 단위가 cm가 아니라 pt라는 점이다. 습관적으로 cm를 생략하는 Ti*k*Z 때의 버릇이 이상한 결과를 가져올 수 있으므로 주의하여야 한다.
l3draw를 쓸 때의 최대 장점은 익숙한 expl3의 반복문을 제한없이 쓸 수 있다는 것이다. l3draw는 앞으로 기능이 추가되거나 이미 있는 명령의 인터페이스가 바뀔 수 있다는 점을 알고 있어야 한다. 안정화되면 expl3로 편입될 것이나 언제가 될지는 모른다.
`\color_fill:n` 명령과 함께 쓰이고 있는 color에 대하여 보충설명을 해둔다. 우리는 지금까지 색상을 위하여 xcolor 패키지에 의존해왔지만 expl3 자신이 color를 다루는 방법을 제공하고 있다. 살짝 귀찮아서(?!) 그냥 친숙한 xcolor로 만족하고 있기는 하나 expl3의 color에 대해서도 알아두는 것이 좋을지도 모르겠다.

```
\ExplSyntaxOn
\color_set:nnn { mycolor } { rgb } { 0.1, 0.8, 0.2 }
\hbox_set:Nn \l_tmpa_box { \color_select:n { mycolor } \rule { 10pt } { 10pt
 ↪  } }
\box_use:N \l_tmpa_box
\ExplSyntaxOff
```

위의 코드가 expl3의 color 관련 명령의 예제이다.[4]
다음 "이름붙인 색상"은 그 값을 바꿀 수 없다.

```
black, white, red, green, blue, cyan, magenta, yellow
```

색상 표현을 위해서는 둘 이상의 색상을 섞을 수 있다.

```
\ExplSyntaxOn
\color_select:n { red!10!magenta!80!yellow }
\rule{20pt}{10pt}
\ExplSyntaxOff
```

`\color_set:nn`이라는 두 개의 인자만을 취하는 명령으로 이 새로운 색상에 이름을 부여할 수 있다.

```
\ExplSyntaxOn
\color_set:nn { fancyyellow } { red!20!magenta!30!yellow }
\color_select:n { fancyyellow } \rule{20pt}{10pt}
\ExplSyntaxOff
```

색상모델을 이용하여 색을 정의할 수 있다. 현재 사용할 수 있는 색상모델은 gray (grayscale), cmyk, rgb, spot 네 가지인데 마지막 spot은 별색을 위한 것이므로 일단 논외로 하면 cmyk와 rgb가 제공되는 것으로 보면 되겠다. rgb로 색정의하는 것을 위에서 이미 보였다.
요컨대, (비록 experimental이기는 하지만) expl3의 기본 기능으로 색상과 그리기가 가능하다는 것이다. (물론 xcolor와 Ti*k*Z를 잘 사용하는 방법을 익혀두는 것이 먼저이다.)
색상의 할당은 원칙적으로 local인데 색상이 유효한 scope를 특별히 취급하기 위하여

[3] 2020/02. 인자를 주는 순서가 바뀌었다.
[4] `l3color`는 2021년에 EXPL3의 일부로 편입되었다.

`\color_group_begin:`
`\color_group_end:`

라는 명령이 주어져 있고, 또한 box를 위해서

`\color_ensure_current:`

라는 명령이 정의되어 있는데, box에 대해서 학습할 때 한 번 정도 나올 것이다.
`\color_fill:n` 명령은 l3draw에서 색상을 활용하게 한다.

사족. 현재 l3color의 도움문서를 보려면 `texdoc l3color` 명령으로 안 될 수 있다. 이 명령은 interface3를 열어줄지 모른다. 그럴 경우라면

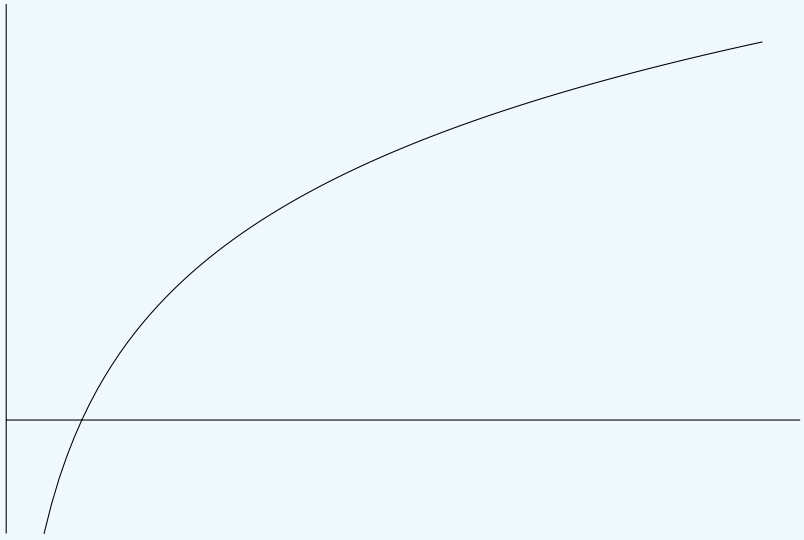
`texdoc l3color.pdf`

와 같이 확장자까지 지시하여 원하는 문서를 열 수 있다. 이렇게 된 이유는 l3color의 일부가 expl3로 들어온 부분이 있기 때문이다.

보조문제

0.5부터 10까지 상용로그 함수를 플로팅하여라.

```
\ExplSyntaxOn
\draw_begin:
    \draw_path_moveto:n { 0,0 }
    \draw_path_lineto:n { 10.5cm, 0cm }
    \draw_path_moveto:n { 0,-1.5cm }
    \draw_path_lineto:n { 0cm, 5.5cm }
    \draw_path_moveto:n { 0.5cm, 5*(ln(0.5)) / ln (10) cm}
    \int_step_inline:nnn { 5 } { 100 }
    {
        \fp_set:Nn \l_x_fp { #1 /10 }
        \fp_set:Nn \l_y_fp { 5 * ( ln (#1/10) / ln (10) ) }
        \draw_path_lineto:n { \l_x_fp cm, \l_y_fp cm }
    }
    \draw_path_use_clear:n { stroke }
\draw_end:
\ExplSyntaxOff
```



그래프를 예쁘게 그리려면 축의 모양, 화살표, 축의 tick 모양, 텍스트 등이 적절하게 어우러져야 한다. 이런 "번거로운" 문제를 일절 생략하고 단지 축과 그래프 자체만 그려보겠다.
x축과 y축이 "축"인 것처럼 보이려면 화살표가 필요한데 l3draw는 (그 목적에 걸맞게) 이런 라이브러리를 갖추고 있지 않다. 만약 정 필요하다면 화살표를 그리는 함수들을 (지금으로서는) 자신이 작성하여야 한다. 눈금과 숫자 같은 것은 어렵지 않지만 코드를 읽기 어렵게 만들 게 뻔해서 모두 생략하였다.
그래프가 취하는 값은 우리가 지난번에 그려본 상용로그 값의 플로팅 때와 마찬가지로 y축을 5배 확대하여 표현한다.

연습문제

기본 정n각형의 한 내각의 크기가

$$180^\circ \times \frac{n-2}{n}$$

임을 유치원에서 배웠다. 인자로 3 이상의 자연수를 받아들여서 정n각형을 그려주는 명령을 l3draw로 작성하여라. 한 변의 길이는 1cm로 한다. 다각형의 내부를 칠하여도 좋다.

입력: `\regpoly{3}`

출력: 

Nova de Hi.

[No. 201] **99 bottles of beer** ____년 __월 __일

문제

"99 bottles of beer"를 expl3로 출력하여라.

초급 프로그래밍 예제로 유명한 "99 bottles of beer"는 다음과 같은 노래를 출력하라는 것이다.

> 99 bottles of beer on the wall, 99 bottles of beer.
> Take one down and pass it around, 98 bottles of beer on the wall.
>
> 98 bottles of beer on the wall, 98 bottles of beer.
> Take one down and pass it around, 97 bottles of beer on the wall.
>
> [……]
>
> 1 bottle of beer on the wall, 1 bottle of beer.
> Take one down and pass it around, no more bottles of beer on the wall.
>
> No more bottles of beer on the wall, no more bottles of beer.
> Go to the store and buy some more, 99 bottles of beer on the wall.

다음 조건을 잘 구현하여야 한다.

(1) 숫자(병의 수)는 99에서 1씩 줄어든다. 마지막은 0이고 다시 99로 돌아온다.
(2) 한 병이 줄면 숫자를 1 줄여서 적어야 한다.
(3) 1일 때는 "bottle"이고 그밖의 경우에는 "bottles"이다.
(4) 0일 때는 "no more bottles"이다.
(5) 0이 되면 두 번째 행이 달라진다.

이를 각종 프로그래밍 언어로 구현해둔 사이트 `http://www.99-bottles-of-beer.net/`에 가보면 수많은 언어의 예제를 볼 수 있다. 다음은 Basic 아류인 Gambas 샘플이다.

```
STATIC PUBLIC SUB Main()
DIM X AS Integer
DIM ProcStr AS String
FOR X=99 TO 0 STEP -1
ProcStr=X & " bottles of beer"
IF X=1 THEN ProcStr="1 bottle of beer"
IF X=0 THEN ProcStr="No more bottles of beer"
IF X<>99 THEN PRINT "Take one down and pass it around, " & ProcStr & " on the
 ↪  wall."
PRINT ProcStr & " on the wall, " & ProcStr &"."
NEXT
PRINT "Go to the store and buy some more, 99 bottles of beer on the wall."
END
```

경우를 나누어서 생각해보면,

1. 카운터가 99일 때: "99 bottles"로 시작하는 첫 행을 적고, "Take one down and pass it around,"까지 인쇄한다.

```
99 bottles of beer on the wall, 99 bottles of beer.\\
Take one down and pass it around,
```

2. 카운터가 98부터 2까지일 때: 먼저 "*n* bottles of beer on the wall." 을 한 번 적고 개행한 다음에, 이것을 두 번 더 반복한다. 그리고 "Take..." 를 적는다.

```
                    98 bottles of beer on the wall. \par
98 bottles of beer on the wall, 98 bottles of beer.\\
Take one down and pass it around,
```

3. 카운터가 1일 때, "1 bottle of ..." 를 한 번 적고 행을 바꾼 다음 다시 두 번 더 적고, "Take..." 를 인쇄한다.

```
                  1 bottle of beer on the wall. \par
1 bottle of beer on the wall, 1 bottle of beer.\\
Take one down and pass it around,
```

4. 카운터가 0일 때. "no more bottles of" 를 한 번 적고 행을 바꾼 다음, 다시 두 번 더 적고, "Go to the store" 로 시작하는 마지막 행을 적은 후, 개행하고 종료한다.

```
                          no more bottles of beer on the wall. \par
No more bottles of beer on the wall, no more bottles of beer.\\
Go to the store and buy some more, 99 bottles of beer on the wall.
```

문서 명령을 `\PrintSong`이라고 하자. 옵션 인자를 하나 받아서 최대 몇 병인지로 삼고 옵션이 없으면 99로 한다.

```
\ExplSyntaxOn
\int_new:N \g_maxbottle_int
\NewDocumentCommand \PrintSong { O{99} }
{
    \int_gset:Nn \g_maxbottle_int { #1 }
    \int_step_function:nnnN { \g_maxbottle_int } { -1 } { 0 }
     ↪  \print_the_song:n
}
```

공통적으로 쓰이는 문자열을 매크로로 정의해둔다.

```
\tl_set:Nn \ofbeer { of~beer }
\tl_set:Nn \onthewall { on~the~wall }
\tl_set:Nx \ofbeeronthewall { \ofbeer{}~\onthewall }
\tl_set:Nn \takeone { Take~one~down~and~pass~it~around }
\tl_set:Nn \gotothestore { Go~to~the~store~and~buy~some~more }
```

이제 `\print_the_song:n`이라는 함수를 정의한다. 인자는 `\int_step`문으로부터 넘어오는 인덱스 카운터이다.

```
\cs_new:Npn \print_the_song:n #1
{
    \int_case:nnF { #1 }
    {
        { 1 } { 1~bottle~\ofbeeronthewall . \par
                1~bottle~\ofbeeronthewall , ~ 1~bottle~\ofbeer . \\
                \takeone,~
              }
```

```
        { 0 } { no~more~bottles~\ofbeeronthewall . \par
                No~more~bottles~\ofbeeronthewall , ~ no~more~bottles~\ofbeer
                 ↪  . \\
                \gotothestore,~\int_use:N
                 ↪  \g_maxbottle_int{}~bottles~\ofbeeronthewall .
              }
        { \g_maxbottle_int } {
                #1~bottles~\ofbeeronthewall , ~ #1~bottles~\ofbeer. \\
                \takeone,~
              }
    }
    {
        #1~bottles~\ofbeeronthewall . \par
        #1~bottles~\ofbeeronthewall , ~ #1~bottles~\ofbeer . \\
        \takeone,~
    }
}
\ExplSyntaxOff
```

본문에서는 이 명령을 verse 환경 안에 넣으면 된다. 99는 너무 많고 지루하니 5개만 보이기로 한다.

```
\begin{verse}
\PrintSong[5]
\end{verse}
```

5 bottles of beer on the wall, 5 bottles of beer.
Take one down and pass it around, 4 bottles of beer on the wall.

4 bottles of beer on the wall, 4 bottles of beer.
Take one down and pass it around, 3 bottles of beer on the wall.

3 bottles of beer on the wall, 3 bottles of beer.
Take one down and pass it around, 2 bottles of beer on the wall.

2 bottles of beer on the wall, 2 bottles of beer.
Take one down and pass it around, 1 bottle of beer on the wall.

1 bottle of beer on the wall, 1 bottle of beer.
Take one down and pass it around, no more bottles of beer on the wall.

No more bottles of beer on the wall, no more bottles of beer.
Go to the store and buy some more, 5 bottles of beer on the wall.

연습문제

만약 `\return_a_verse:n`이라는 함수가 있어서, 인자로 주어지는 수에 해당하는 연을 출력하도록 만들고 싶다. 이 함수를 정의하고 이를 이용하여 위의 문제를 다시 해결하되, 1은 "one" 으로 나타내어라. 예컨대, `\return_a_verse:n { 5 }`의 결과는

> 5 bottles of beer on the wall, 5 bottles of beer.
> Take one down and pass it around, 4 bottles of beer on the wall.

로 출력된다.

문제

두 개의 박스를 잇대어서 왼쪽에는 LaTeX 소스의 입력을 오른쪽에는 그 출력 결과를 보여주는 `myexample` 환경을 만들어보아라.

28 file

LaTeX의 성공 요인은 여러 가지가 있지만 맨처음 등장했을 때 사람들을 매료했던 것이 목차와 상호참조의 자동생성 기능이었다. 세상에 태어나니까 이미 Microsoft Word가 존재했던 사람들이야 이해하기 어렵겠지만 이 때가 1980년대였음을 상기하자.

28.1 aux, toc, lot, lof라는 파일

`latex`(앞으로 `xelatex`이나 `pdflatex` 등 실행 명령을 가리킬 때 `latex`이라고 지칭한다)을 실행하였을 때 여러 가지의 부수파일이 만들어지는데 그 가운데 확장명이 `.aux`인 것이 있다. `latex`이 실행하면서 이런저런 정보들을 적어둔 파일이다. 여기에는 장절의 번호, 표제, figure/table의 캡션과 번호, `\label`들에 대한 정보 등을 담고 있다. 다음 번 `latex` 실행 시에 이 정보를 읽어와서 참조와 (hyperref의 도움을 받아서) 하이퍼링크 등을 만들어낸다. table of contents는 aux 이외에 `toc`와 `out`이라는 다른 파일을 더 이용하므로 (`out`은 주로 pdf bookmark를 만드는 데 쓴다) aux만으로 이루어지는 것은 아니지만 관련된 정보는 담고 있다. `\includeonly` 명령을 써서 chapter 2만 조판하려 할 적에 조판되지 않는 chapter 1의 마지막 페이지 번호 같은 것은 aux에서 가져오는 것이다. (따라서 chap1.aux라는 파일이 없으면 2장의 시작 페이지가 제대로 조판되지 않고 chapter 1에 있는 참조 정보도 활용할 수 없다.)
그러니 `.aux` 파일은 웬만하면 지우지 말자. 실행 오류 때문에 초기화 컴파일이 필요할 때를 제외하고.
`toc`, `lof`, `lot` 확장자를 갖는 파일들은 각각 목차, 그림 목차, 표 목차를 넣어두고 있는 파일들이다. `\tableofcontents` 같은 명령이 불릴 적에 여기에 기록된 정보를 가지고 목차를 만든다.
이 파일들에 뭔가를 임의로 적어넣는 것이 가능하다. 아무거나 적어넣으면 이상한 결과를 보게 될 것인데 예컨대 `toc` 파일은

```
\contentsline {section}{Introduction}{1}{section.0.1}
```

과 같은 행들로 이루어져 있다. 이 파일과 상호작용하기 위한 명령 `\addtocontents`나 `\addcontentsline` 따위가 미리 정의되어 있고 이에 대한 설명은 예컨대 memoir 매뉴얼에 상세하다. 그러므로 이 명령이나 파일들에 대해서 여기서 더 말하지 않으려 한다.
외부 파일을 이용하는 LaTeX의 기능으로 index(색인)과 bibliography가 더 있다. index에 대해서만 말해보자면, `\index`라는 명령이 주어지는 순간에 그 인자와 현재 위치를 `.idx`라는 파일에 쓴다. 한 번 컴파일이 끝나고 나면 `.idx` 파일에는 그 소스에 나온 모든 `\index` 엔트리들을 저장하고 있다.
`makeindex`라는 프로그램은 `.idx`를 읽어서 일단 정렬하고 식자할 수 있는 형태로 만들어서 `.ind` 파일에 저장한다. 이 때에 "스타일"이라고 하는 것을 적용하는데 이 스타일 파일은 `makeindex`가 이해하는 언어로 작성되어 있어야 한다. 보통 `.ist`라는 확장명을 갖는다. 아무튼 이 과정은 `latex`이 관여하는 것이 아니다.
`latex`은 만약 `.ind` 파일이 있으면 그 내용 전체를 `\printindex` 명령이 불린 위치에 그대로 삽입한다.
`komkindex`는 한글 UTF-8 문자열도 처리가 가능하도록 한 `makeindex` wrapper이다. 한글이 문제가 되면 `komkindex`를 쓰는 이유는 거기에 있다. 최근 `makeindex`를 대체하기 위한 노력들, 예컨대 `xindy`, `xindex` 등이 개발되는 중에 있지만 핵심 개념은 이에서 벗어나지 않는다.

28.2 verbatim이라는 것

verbatim [vəːrbéɪtəm]은 lshort을 읽었다면 다 알고 있을 것으로 본다. LaTeX에 관한 LaTeX 문서를 작성하려면 부득이하게 verbatim으로 떡칠(...)된 소스를 만들 수밖에 없다. verbatim에 대하여 반드시 기억해두어야 할 것은, 이것이 "more fragile than fragile" 이라는 사실이다. verbatim 텍스트는 일단 verb 상태가 되고 나면 다른 명령의 인자로 사용될 수 없다. 각주에도 들어갈 수 없고, 장절 명령과 같은 moving arguments에도 들어가지 못한다. verbatim 내부에서 명령을 통한 조작을 가할 수도 없다. 그래서 일찍부터 verbatim을 좀더 fancy하게 다루는 수많은 방법이 개발되어 있다. minted 패키지가 사용하는 fancyvrb 같은 것이 대표적이다. 아마도 LaTeX 패키지 가운데 verbatim 관련이 종류가 많고 비슷한 것들도 많은 부류에 속할 것이다. 주로 Source Code Listing 관련하여 많은 요구가 있어왔기 때문이다. 이 패키지들은 위에 열거한 verbatim의 제약을 조금이라도 벗어나기 위한 몸부림이다.
expl3의 입장에서 말하자면 verbatim은 string과 비슷하다. 예를 들어

```
\ExplSyntaxOn
\ttfamily
\tl_to_str:n { \verbatim \l_tmpa_tl }
\ExplSyntaxOff
```

```
\verbatim \l_tmpa_tl
```

이 예를 보면 typewriter font로 `\tl_to_str:n`한 결과를 출력하면 그것이 verbatim이라는 것이다. 우리는 expl3의 막강한 arguments expansion 함수들을 이용하여 원하는 결과를 출력할 수 있다.

```
\ExplSyntaxOn
\ttfamily
\tl_set:Nn \l_tmpa_tl { And~silent~be }
\tl_to_str:n { \l_tmpa_tl } \\
\exp_args:No \tl_to_str:n { \l_tmpa_tl }
\ExplSyntaxOff
```

```
\l_tmpa_tl
And silent be
```

그러나 expl3라 하더라도 verbatim 자체는 주의깊게 다루어야 한다는 사실을 기억하자. xparse의 v형 인자는(expl3의 cs인자형의 v와는 다른 것이다) 다른 명령이나 함수에 인자로 넘겨주지 못한다는 제한은 이래서 생겨났다.

28.3 example 환경 설계의 아이디어

문제로 돌아와서 생각해보자. 한 번은 입력 형태이고 한 번은 조판된 형태를 보여야 하니까 왼쪽 박스에는 verbatim 텍스트가, 오른쪽 박스에는 그 조판 형태가 나와야 한다. 이것은 같은 소스를 한 번은 verbatim으로 다른 한 번은 입력 코드로 두 번을 읽어야 한다는 뜻이다.
단순, 명쾌, 무식하게 두 번 입력하여 이런 비슷한 것을 만드는 것을 먼저 해보려 한다.

```
\begin{minipage}{.5\textwidth}
\begin{verbatim}
\begin{verse}
Look, stranger, on this island now \\
The leaping light for your
   delight discovers, \\
Stand stable here \\
And silent be, \\
```

```
That through the channels
   of the ear \\
May wander like a river \\
The swaying sound of the sea.
\end{verse}
\end{verbatim}
\end{minipage}%
\begin{minipage}{.5\textwidth}
\begin{verse}
Look, stranger, on this island now \\
The leaping light for your
delight discovers, \\
Stand stable here \\
And silent be, \\
That through the channels of the ear \\
May wander like a river \\
The swaying sound of the sea.
\end{verse}
\end{minipage}
```

```
\begin{verse}
Look, stranger, on this island now \\
The leaping light for your
   delight discovers, \\
Stand stable here \\
And silent be, \\
That through the channels
   of the ear \\
May wander like a river \\
The swaying sound of the sea.
\end{verse}
```

Look, stranger, on this island now
The leaping light for your delight
discovers,
Stand stable here
And silent be,
That through the channels of the ear
May wander like a river
The swaying sound of the sea.

똑같은 텍스트를 두 번 쓰면 문제가 되는 것이 귀찮을 뿐더러 둘 중 하나만 나중에 수정하게 될 수도 있다는 점이다. 단일 소스를 두 번 읽을 수 있으면 좋겠다. 여기 식자할 내용 전체를 외부의 어떤 파일에 write해두었다가 그 파일을 mode를 달리하여 두 번 읽으면 되지 않을까?

28.4 input과 include

이미 파일이 있다고 하자. 이것을 읽어오는 방법은 무엇인가?
`\input`이라는 primitive는 외부 파일을 통째로 읽어와서 그 명령이 불린 위치에 그대로 삽입해준다. verbatim으로 읽어오는 것이 아니라는 점을 알아두어야 한다. 매크로는 들어와서 그대로 실행되고 주석문은 무시된다.
참고로 LaTeX의 `\include`라는 명령이 있는데 이것은 용도가 완전히 다르다. 이것은 주로 긴 LaTeX 문서를 부분별로 작성하려 할 때 쓰는 것으로 `\include`되는 파일은 `\documentclass`가 없는 파일이어야 하고 문서의 본문에서 불려야 하며 부수 정보(`.aux`)를 함께 불러오고 항상 새로운 페이지로 시작한다. `\includeonly`는 본문에 출현할 `\include` 명령을 일부 활성화/무시할 수 있게 해주는 것이다. 당연히 원시 명령인 `\input`에는 그런 거 없다.
어디에 있는 파일을 불러올까? 오늘날 우리가 쓰는 TeX Live와 같은 텍 시스템은 `kpathsea`라는 라이브러리에 기대어 컴파일되어 있다. 이 말은 `kpathsea`가 그 위치를 알려줄 수 있는 파일이면 모두 `\input`할 수 있다는 의미이다. 현재의 작업 폴더와 그 상대위치로 지정된 파일, `TEXMF/tex` 위치 아래의 파일은 다른 조치없이 바로 `\input` 가능하다. 상대 위치를 path로 주는 경우에 Windows 시스템이라도 path 경로는 반드시 `/`로 주어야 한다. 작업 폴더 아래 `texts`라는 하위 폴더를 만들고 그 안에 `testa.tex`이라는 파일이 있을 때,

```
\input{texts/testa.tex}
```

으로 하면 운영체제에 상관없이 모두 잘 불러올 것이다.
plain TEX의 \input 명령과 LATEX의 \input 명령은 거의 같지만 한 가지가 다른데 LATEX의 것이 "중괄호 인자"를 쓸 수 있다는 것이다.

```
\input inputfile.tex
\input{inputfile.tex}
```

아래쪽 것이 LATEX 방식이다. 이 LATEX 명령의 편리한 점 하나는 확장자가 .tex인 파일은 파일 이름만 써도 된다는 것이다.

28.5 verbatiminput

그런데 \input은 파일의 contents를 "입력파일"로 읽어온다. 이래서는 verbatim으로 표현할 수가 없다. 심지어

```
\verb+\input{auden.tex}+
```

이런 명령은 어떤 방법으로도 auden.tex이라는 파일의 내용을 \verbatim의 인자로 만들 수 없다. 위의 것은 그냥 \input{auden.tex}이라는 글자를 찍어줄 뿐이다.
이 문제 해결에 관련된 기나긴 역사를 요약하려면 별도의 글이 필요하다. 우리는 이왕에 memoir 클래스를 쓰고 있으므로 memoir가 이미 이 문제를 간단히 해결해두고 있다는 점을 지적하고 지나가려 한다.

```
\verbatiminput
\boxedverbatiminput
```

이에 대한 상세한 설명은 memoir 매뉴얼을 보자.

```
\boxedverbatiminput{auden.tex}
```

```
\begin{verse}
Look, stranger, on this island now \\
The leaping light for your
delight discovers, \\
Stand stable here \\
And silent be, \\
That through the channels of the ear \\
May wander like a river \\
The swaying sound of the sea.
\end{verse}
```

28.6 잠정적 해결

```
\ifvmode\leavevmode\fi
\begin{minipage}{.5\textwidth}
\verbatiminput{auden.tex}
\end{minipage}
\begin{minipage}{.45\textwidth}
\input{auden.tex}
\end{minipage}
```

```
\begin{verse}
Look, stranger, on this island now \\
The leaping light for your
delight discovers, \\
Stand stable here \\
And silent be, \\
That through the channels of the ear \\
May wander like a river \\
The swaying sound of the sea.
\end{verse}
```

Look, stranger, on this island now
The leaping light for your delight discovers,
Stand stable here
And silent be,
That through the channels of the ear
May wander like a river
The swaying sound of the sea.

`\boxedverbatiminput`은 다른 환경 안에 들어가기 어렵다. 그래서 `\verbatiminput`만을 썼는데 일단 소스와 결과를 나란히 놓는 것까지는 어떻게든 해본 셈이다.
이제 주어진 텍스트를 어떻게 "임의의 파일"에 쓸 것인가가 남았다.

29 외부 파일에 쓰기

29.1 stream과 file

파일 조작의 기본은 다음과 같다.

(1) 파일을 연다. 이것은 운영체제에게 특정 (물리)파일의 내용을 전달해달라고 요청하는 것이다. 이렇게 해서 전달받은 파일의 내용을 *stream*에 연결해둔다. 열기는 "읽기 위해 열기"와 "쓰기 위해 열기"가 있다.

(2) `latex`은 오로지 stream만을 조작한다. "쓰기 위해 연" stream에는 아무 것도 없는 것이 기본이므로 만약 파일 내용을 수정하려면 "읽기 위해 열고" 그 내용을 (텍스트 파일에서 주로 한 행씩) 수정하여 "쓰기 stream"으로 보내어야 한다.

(3) stream 조작이 끝나면 이 파일을 닫는다. "쓰기 위해 연" stream은 파일이 닫히는 순간 OS는 stream의 내용을 그대로 물리 파일로 쓴다.

파일 조작에서 가장 중요한 것은 열고 닫는 것이다. 열린 stream은 반드시 닫아주어야 한다. `latex`은 실행을 시작할 때 main aux라고 부르는 `.aux`를 "읽기 위해" 연다. 그리고 같은 파일을 "쓰기 위해" 열고 컴파일이 끝나면 이것을 닫는다. 즉, 기존에 `.aux`에 쓰여 있던 내용들이 "읽기"로 모두 들어오고, 컴파일하면서 생성되는 정보는 "쓰기"가 이루어진다. "쓰기"로 읽었기 때문에 이 파일 자체는 이전 내용과 상관없이 완전히 새롭게 쓰여지는데 이전 내용이 이미 읽어진 후이기 때문에 "이전 내용이 반영되어 변경되는 것처럼" 보이는 것이다.
"즉시 쓰기" (immediate write)라는 것이 있다. 이것은 (그 레지스터 번호를 사용자가 알 필요가 없는) 어떤 stream으로 주어진 file을 연결하여 열고 주어진 인자를 그 stream으로 보낸 다음 즉시 file을 close하는 일을 한다. stream이 열려 있지 않고 새로이 열고 바로 닫는 것이기 때문에 이 명령이 여러 번 불리면 그 때마다 파일 내용이 변경되어 이전 내용이 보존되지 않는다.
원칙적으로 stream으로 보내기는 other로 이루어진다. 즉 verbatim으로 보내는 것이다. 그러나 LaTeX이 stream으로 보내면서 "매크로가 확장된 이후에" 보내면 명령이 풀리거나 실행된 상태가 저장될 것이고 "매크로를 확장하지 않고 있는 그대로" 보내면 verbatim 쓰기가 될 것이다. "쓰기"에서는 항상 "언제 확장할 것인가"가 매우 중요하다.

29.2 filecontents 환경

예전부터 사용된 "파일 쓰기" 대표적인 것이 filecontents 환경이다. LaTeX 자체가 제공하는 환경이고 오랜 역사를 자랑한다. 이 환경은 \documentclass 명령 이전에 써야 하고 파일 이름을 인자로 줄 수 있으며 환경 안의 내용이 verbatim으로 저장된다. 만약 같은 이름의 파일이 존재한다면 아무 일도 일어나지 않으므로 새 파일을 만드는 것만 가능하다. 한편 별표를 붙인 환경은 이 파일에 붙는 간단한 기록 정보를 나타내는 주석문(%로 시작하는 행)을 제외하고 쓴다.
이 환경은 주로 한 개의 파일에 스타일 파일을 같이 넣어서 전달하거나 할 때 쓰였다.
위에 언급한 제약을 조금 완화하여 본문 어디에서나 사용할 수 있도록 수정한 filecontents 패키지가 있다. 원래 아래 보기는 \documentclass 이전에 두어야 하지만 이 패키지의 도움을 받아 여기 예제를 작성해 본다.

```
\begin{filecontents}{mytest1.tex}
\begin{verse}
Look, stranger, on this island now \\
The leaping light for your delight discovers, \\
Stand stable here \\
And silent be, \\
That through the channels of the ear \\
May wander like a river \\
The swaying sound of the sea.
\end{verse}
\end{filecontents}
```

현재 폴더에 mytest1.tex이 생겨나 있는지 확인하여라. 이것을 \boxedverbatiminput하면

```
%% LaTeX2e file `mytest1.tex'
%% generated by the `filecontents' environment
%% from source `e3sg-full' on 2020/09/17.
%%
\begin{verse}
Look, stranger, on this island now \\
The leaping light for your
   delight discovers, \\
Stand stable here \\
And silent be, \\
That through the channels
   of the ear \\
May wander like a river \\
The swaying sound of the sea.
\end{verse}
```

위와 같이 된다. 처음의 주석문 4줄을 붙이지 않으려면 filecontents* 환경 안에 넣는다.
예전부터 전해내려오는 주의사항 중에 "filecontents의 인자인 파일 이름을 현재 작성중인 문서 파일의 이름과 같게 하지 말라"는 것이 있다. 생각해보면 어차피 filecontents 환경은 같은 이름의 파일이 있으면 아무 일도 하지 않으니까 결과적으로 아무 일도 일어나지 않겠지만 코딩 오류인 것은 틀림없다.

29.3 memoir의 file 관련 명령

memoir는 파일을 다루는 명령이 기본적으로 제공한다. 관련 내용은 매뉴얼을 참고하고 우리가 필요한 부분만 살펴보겠다. 특히 읽어오기에 관한 부분은 생략한다.

```
\newoutputstream{<stream>}
\openoutputfile{<filename>}{<stream>}
\addtostream{<stream>}{text}
\closestream{<stream>}
```

설명이 필요없을 것이다. \addtostream 명령은 text로 오는 부분을 "미리 확장해서" stream으로 보낸다.

```
\newoutputstream{outputone}
\openoutputfile{mytest2.tex}{outputone}
\def\myname{Nova de Hi.}
\addtostream{outputone}{\myname}
\closeoutputstream{outputone}

\verbatiminput{mytest2.tex}

\fbox{\input{mytest2.tex}}
```

```
Nova de Hi.
```

Nova de Hi.

반면, 일절 확장하지 않고 verbatim으로 쓰는 데는 두 가지 방법이 있는데 하나는 열려 있는 stream에 내용을 추가해갈 수 있는 writeverbatim (stream) 환경이고 다른 하나는 즉시 쓰기를 실행하는 verbatimoutput (filename)이다. 따라서 verbatimoutput 환경의 경우 다음 번에 실행되면 이전 내용은 사라진다.
memoir뿐 아니고 tcolorbox가 이용하고 있는 listings나 sverb example-p 등의 비슷한 기능을 제공하는 패키지들이 모두 verbatim으로 쓰고 읽기를 지원하고 있다.
memoir의 file 관련 명령을 이용하여 원래 우리가 하고자 했던 것을 해보자.

```
\newoutputstream{myout}
\openoutputfile{\jobname-test.tex}{myout}

\begin{writeverbatim}{myout}
\begin{verse}
Look, stranger, on this island now \\
The leaping light for your
delight discovers, \\
Stand stable here \\
And silent be, \\
That through the channels of the ear \\
May wander like a river \\
The swaying sound of the sea.
\end{verse}
\end{writeverbatim}

\closeoutputstream{myout}

\noindent
\begin{minipage}{.5\textwidth}
\verbatiminput{\jobname-test.tex}
\end{minipage}%
\begin{minipage}{.5\textwidth}
\input{\jobname-test.tex}
\end{minipage}
```

```
\begin{verse}
Look, stranger, on this island now \\
The leaping light for your
delight discovers, \\
Stand stable here \\
And silent be, \\
That through the channels of the ear \\
May wander like a river \\
The swaying sound of the sea.
\end{verse}
```

Look, stranger, on this island now
The leaping light for your delight
discovers,
Stand stable here
And silent be,
That through the channels of the ear
May wander like a river
The swaying sound of the sea.

30 expl3의 file

expl3는 단순히 verbatim을 읽고 쓰기 위한 환경들과는 달리 좀더 섬세하게 파일을 조작할 수 있도록 해준다. expl3에서 파일 관련 제공하는 것은 세 종류가 있다. 하나는 파일로부터 "읽어오기" 위한 함수들로서 `\ior_...`로 시작한다. 다른 하나는 파일에 "쓰기" 위한 함수들로서 `\iow_...`로 시작한다. 그리고 `\file_...`은 파일의 경로, 존재여부, 비교 등을 위한 함수를 제공한다.
파일 읽기에 대하여 조금 말해둔다. `\input{<filename>}`과 같은 방법으로 파일이 존재하면 그 내용을 전부 input하는 것은

```
\file_input:n { <filename> }
\file_if_exist_input:n { <filename> }
```

이 있다. 그런데 `\ior_...` 읽기라는 것은 이처럼 파일 전체를 통째로 가져오는 것이 아니라 스트림으로부터 한 줄씩 처리하는 것을 기본으로 하는 것이다.
"읽기"를 위한 `ior`과 "쓰기"를 위한 `iow`를 다음과 같이 선언하는 것은

```
\ior_new:N
\iow_new:N
```

읽거나 쓰기 위한 스트림을 선언하는 것이다. (file에는 `\l_tmpa_ior` 같은 스크래치 변수가 없다는 사실에 주의하자.)
이 스트림에 파일을 연결하여 여는 것은

```
\ior_open:Nn \l_inputfile_ior { file-to-read.txt }
\iow_open:Nn \l_outputfile_iow { file-to-write.txt }
```

이렇게 한다.
파일 작업이 종료되면 이를 닫아야 한다.

```
\ior_close:N
\iow_close:N
```

"읽기"를 위한 조작을 보자. 기본적으로 한 줄(line)씩 읽어온다.

```
\ExplSyntaxOn
\ior_new:N \l_test_ior
\ior_open:Nn \l_test_ior { \jobname-test.tex }
\ior_get:NN \l_test_ior \l_tmpa_tl
\ExplSyntaxOff
```

앞서 저장한 파일 \jobname-test.tex에서 처음 한 줄을 "입력 스트림으로 읽어서" \l_tmpa_tl에 저장하였다. 어떻게 읽느냐는 아주 중요한데 \ior_get:NN 함수는 파일에 있는 토큰들을 normal token으로 읽는다. 그러나 \ior_str_get:NN은 "string으로 (verbatim으로) 읽어서" \l_tmpa_tl에 저장한다. _str_가 붙은 함수는 모두 verbatim으로 읽는 것이다.
\ior_map_inline:Nn은 파일을 끝까지 읽을 수 있게 한다. #1은 현재 읽은 한 줄이다. string으로 읽는 대응 함수 \ior_str_map_inline:Nn이 있다. 파일 읽기를 중간에 중단하려면 \ior_map_break:를 쓴다.
\ior_if_eof:NTF는 입력 스트림의 끝인가를 검사하는 것이다.

```
\ExplSyntaxOn
\ior_new:N \l_test_ior
\ior_open:Nn \l_test_ior { \jobname-test.tex }
\ior_str_map_inline:Nn \l_test_ior
{
    \fbox { \sffamily\small \color{blue} #1 } \par
}

\ior_close:N \l_test_ior
\ExplSyntaxOff
```

```
\begin{verse}
Look, stranger, on this island now \\
The leaping light for your
delight discovers, \\
Stand stable here \\
And silent be, \\
That through the channels of the ear \\
May wander like a river \\
The swaying sound of the sea.
\end{verse}
```

쓰기 위하여 스트림을 여는 것은 앞서 보았다. 실제로 내용을 써넣는 것이 중요한데,

\iow_now:Nn
\iow_now:Nx

이 둘 사이의 차이를 이해하는 것이 중요하다. :Nx 쪽은 매크로가 있다면 확장한 다음에 스트림으로 보낸다.

```
\ExplSyntaxOn
\iow_new:N \l_tmp_iow
\iow_open:Nn \l_tmp_iow { \jobname-testa.txt }
\iow_now:Nn \l_tmp_iow { \l_tmpa_tl{}~is~a~scratch~tl. }
\iow_close:N \l_tmp_iow

\verbatiminput{\jobname-testa.txt}

\tl_set:Nn \l_tmpa_tl { <TMP> }

\iow_open:Nn \l_tmp_iow { \jobname-testa.txt }
\iow_now:Nx \l_tmp_iow { \l_tmpa_tl{}~is~a~scratch~tl. }
\iow_close:N \l_tmp_iow
```

```
\verbatiminput{\jobname-testa.txt}

\ExplSyntaxOff
```

```
\l_tmpa_tl {} is a scratch tl.

<TMP>{} is a scratch tl.
```

expl3의 "쓰기"가 verbatim 쓰기와 완전히 같지는 않다. 예를 들면

```
\ExplSyntaxOn
\iow_open:Nn \l_tmp_iow { \jobname-testa.txt }
\iow_now:Nx \l_tmp_iow { \iow_char:N \# \iow_char:N \% }
\iow_close:N \l_tmp_iow
\verbatiminput{\jobname-testa.txt}
\ExplSyntaxOff
```

```
#%
```

와 같이 `\iow_char:N`이나 `\iow_newline:` 처리해야 하는 경우가 있다.
그러므로 파일이나 내용의 일부를 통째로 저장하려 하는 경우에는 expl3의 `\iow`를 쓸 필요없이 그냥 verbatim으로 write하는 편이 쉽다. 그러나 각 변수를 좀더 세밀하게 조작해야 하는 경우라면 이쪽이 훨씬 편리할 때가 있다.
읽기 쪽도 마찬가지라서 파일 전체를 한꺼번에 읽어들이는 일은 `\verbatiminput`과 `\input`이 훨씬 편하다. 그러나 파일 각 행에 일정한 조작을 해야 하는 일이 있다면 expl3가 좋다.
소스와 결과를 나란히 놓는 example 환경은 latexdemo 패키지를 비롯하여 우리가 쓰고 있는 tcolorbox까지 다양한 종류가 이미 있다. 우리는 minipage 두 개를 나란히 두어서 비슷한 일을 해보았지만 예컨대 박스의 높이가 길어질 때 다음 페이지로 나누는 것이라든가 다양한 상황이 존재하기 때문에 이 소박한 minipage 해법으로는 만족스럽지 않을 것이다.

연습문제

기본 13. 명령 `\ListWord`를 정의하려 한다. 이 명령을 단어에 대하여 적용하면 현재 식자되는 위치에서 박스를 친다. 일련번호를 내부적으로 생성하여 기억하고 있다가 문서의 마지막에 List of Words라는 섹션을 만들고 단어의 일련번호, 단어, 발생한 페이지를 나열한다. 마지막의 List of Words 섹션은 두 번째 컴파일 시에 완성되어도 좋다. 이러한 명령 `\ListWord`를 작성하여라. 단어의 정렬은 무시한다.

예제

9pt, 10pt, 11pt, 12pt일 때의 font size 명령으로 1em의 크기가 어떻게 달라지는가를 본 적이 있다. 이것을 현재 작성중인 문서에 하나의 도표로 만들어 넣고 싶다. 어떻게 하면 되겠는가?

31 terminal과 shell

TEX이 이미 정해놓은 특별한 파일이 몇 가지 있다. `terminal` 스트림은 콘솔이다. 터미널에 쓴다는 것은 console로 출력한다는 의미이다. 예를 들어

```
\typeout{Test HERE}
```

이 명령은 컴파일 과정에서 콘솔에 텍스트를 출력해준다. expl3로는

```
\ExplSyntaxOn
\iow_term:n {Test HERE}
\ExplSyntaxOff
```

으로 한다.
한편, 18이라는 레지스터 번호를 가진 스트림이 있다. 이것은 시스템 셸이다. 즉,

```
\immediate\write18{ls -l}
```

이것은 `ls -l`이라는 "명령"을 시스템 콜하는 것이다. 요컨대, 외부 셸 명령을 실행하는 것.
expl3 언어로는 다음처럼 한다.[5]

```
\ExplSyntaxOn
\sys_get_shell:nnN { pwd } {} \l_tmpa_tl
\l_tmpa_tl
\ExplSyntaxOff
```

```
/Users/nova/Documents/testsii/e3sg-2022
```

실제로는 현재 shell escape 값이 1인가를 검사하고 결과가 no value인지를 검사하는 좀더 복잡한 과정이 필요한데 이런 명령을 실용적으로 쓸 일이 많지는 않을 것 같아서 개념만 보였다.
결과를 string으로 받는 경우가 아니라 단순히 실행만 시키면 되는 경우라면

```
\ExplSyntaxOn
\sys_shell_now:n { ls~-l }
\ExplSyntaxOff
```

이렇게 하는데 이것이 `\immediate\write18`에 해당하는 것이다.
예전에는 그랬던 때도 있는데 `latex` 컴파일 과정에서 아무 프로그램이라도 다 실행시킬 수 있게 되고부터 보안 문제가 우려되기 시작했다. "`cd / && rm -rf /`"를 sudo 권한으로 실행하면 곤란하지 않을까?
그래서 현대의 `tex` 관련 실행 파일들은 "제한된 셸 명령 허용" 모드로 실행되게 되어 있다. 아무런 보안 이슈도 발생하지 않는 것이 확실하고 프로그램이 TEX Live 개발자들의 통제 아래 있는 몇 가지 프로그램(`bibtex`,

[5] 2020/02. `\sys_get_shell:nnN`의 이름이 바뀌었다.

makeindex, kpsewhich, r-mpost, repstopdf)만이 허용되도록 되어 있다. pygmentize를 포함해 달라는 요구가 한때 있었던 모양인데 TEX Live 개발팀에서는 filter feature가 insecure하다고 판단하여 제외해두고 있다. 중요한 것은 latex 컴파일 명령 자체는 "안전한 명령"에 들어가지 않는다는 것인데 어떤 프로그램이 시스템 콜로 외부 명령을 실행할 수 있으면 보안상 "위험"으로 분류되므로 그것은 당연하다고 하겠다.
사용자가 자신의 책임 하에 셸 명령을 허용하도록 하는 것이 --shell-escape라는 실행 옵션이다. 지금 이 문서를 컴파일하려면

```
xelatex --shell-escape --synctex=1 esg008
```

이라고 하여야 하는데 minted 패키지가 pygmentize를 실행하여야 하기 때문인 것이다.

32 폰트 사이즈 옵션

임의의 파일을 작성하되 그 파일을 컴파일하면 원하는 정보가 들어 있는 txt 파일을 생성해주도록 해보자. 첫 줄은 다음과 같이 시작해야 할 것이다.

```
\documentclass[9pt]{memoir}
```

여기서 [9pt] 부분은 나중에 10pt, 11pt 등으로 바꿔야 한다. 그리고 expl3를 쓸 수 있도록 해두고 9포인트임을 나타내는 매크로를 하나 정의한다. 이 매크로에는 숫자가 들어가지 않게 하는 것이 중요하다.

```
\usepackage{xparse}
\def\sizeoption{nine}
```

여기 "nine"도 문서 옵션이 달라지면 바꾸어야 할 텍스트이다.

```
\begin{document}
\ExplSyntaxOn
\seq_set_from_clist:Nn \l_fontcmd_seq { miniscule, tiny, scriptsize, small,
 ↪  normalsize, large, Large, LARGE, huge, Huge, HUGE }
\iow_new:N \l_output_iow
\iow_open:Nn \l_output_iow { \jobname-\sizeoption.txt }

\iow_now:Nx \l_output_iow { \use:c { clist_set:cn } { l_ \sizeoption _clist }
 ↪  }
\iow_now:Nx \l_output_iow { \iow_char:N \{ }

\seq_map_indexed_inline:Nn \l_fontcmd_seq
{
    \use:c { #2 }
    \iow_now:Nx \l_output_iow
    {
        \dim_eval:n { 1em }
    }
    \int_compare:nT { #1 < \seq_count:N \l_fontcmd_seq }
    {
        \iow_now:Nx \l_output_iow { , }
    }
}
\iow_now:Nx \l_output_iow { \iow_char:N \} }
\iow_close:N \l_output_iow
\ExplSyntaxOff
\end{document}
```

이렇게 생긴 파일을 생성해야 한다. 바뀌어야 하는 부분이 있으니까 다음처럼 하면 어떨까?
`\begin{document}`부터 끝까지는 공통부분에 해당하고 이 가운데서는 "확장하여 기록해야 할" 부분이 없으며 텍스트의 양이 많으므로 이것을 그냥 verbatim으로 `\jobname-body.tex`이라는 파일에 써두기로 하자. 이 때 `\jobname`은 현재 작성하고 있는 이 문서, 즉 외부에 생성될 문서가 아니라 메인 문서의 이름이다. 여기서는 `esg008`.

```
\begin{verbatimoutput}{\jobname-body.tex}
\begin{document}
\ExplSyntaxOn
\seq_set_from_clist:Nn \l_fontcmd_seq { miniscule, tiny, scriptsize, small,
 ↪  normalsize, large, Large, LARGE, huge, Huge, HUGE }
\iow_new:N \l_output_iow
\iow_open:Nn \l_output_iow { \jobname-\sizeoption.txt }

\iow_now:Nx \l_output_iow { \use:c { clist_set:cn } { l_ \sizeoption _clist }
 ↪  }
\iow_now:Nx \l_output_iow { \iow_char:N \{ }

\seq_map_indexed_inline:Nn \l_fontcmd_seq
{
    \use:c { #2 }
    \iow_now:Nx \l_output_iow
    {
        \dim_eval:n { 1em }
    }
    \int_compare:nT { #1 < \seq_count:N \l_fontcmd_seq }
    {
        \iow_now:Nx \l_output_iow { , }
    }
}
\iow_now:Nx \l_output_iow { \iow_char:N \} }
\iow_close:N \l_output_iow
\ExplSyntaxOff
\end{document}
\end{verbatimoutput}
```

사실 이 부분을 이리 처리하는 데는 다른 이유도 있는데 `\iow_...` 함수를 쓰려면 parameter를 처리하는 것이 상당히 까다로워지기 때문이다. expl3에서 #1은 무조건 제일 먼저 확장되는 부분이므로 위의 것을 `\iow_now:Nn` 하면 #1에서 에러가 나거나 다른 이상한 것이 들어가거나 한다. 이 문제를 피해가는 다른 방법이 있지만 여기서는 더 다루지 않겠다.[6]
그 다음은 다음과 같이 한다.

```
\ExplSyntaxOn

\seq_set_from_clist:Nn \l_tmpa_seq { 9pt, 10pt, 11pt, 12pt }
\seq_set_from_clist:Nn \l_tmpb_seq { nine, ten, eleven, twelve }

\iow_new:N \l_ofile_iow

\cs_new:Npn \gen_files:nn #1 #2
{
    \iow_open:Nn \l_ofile_iow { \jobname-#1.tex }

    \iow_now:Nn \l_ofile_iow { \documentclass [#1] {memoir} }
```

[6] 2022/05. `\seq_indexed_map...` 명령의 이름이 `\seq_map_indexed...`로 바뀌었다.

```
    \iow_now:Nn \l_ofile_iow { \usepackage{xparse} }
    \iow_now:Nn \l_ofile_iow { \def\sizeoption{#2} }
    \iow_now:Nn \l_ofile_iow { \input }
    \iow_now:Nx \l_ofile_iow { \iow_char:N \{ \jobname-body.tex \iow_char:N
     ↪  \} }

    \iow_close:N \l_ofile_iow
}

\seq_mapthread_function:NNN \l_tmpa_seq \l_tmpb_seq \gen_files:nn

\ExplSyntaxOff
```

이 부분은 실제로 파일을 만들어가는 부분이다. 실제로 이 부분이 컴파일되고 나면 esg008-9pt.tex이라는 파일 등이 만들어지며 그 내용은 다음과 같이 되어 있다.

```
\documentclass [9pt]{memoir}
\usepackage {xparse}
\def \sizeoption {nine}
\input
{esg008-body.tex}
```

이것이 위의 코드가 의도하고 있는 바이다.
이런 파일이 9, 10, 11, 12에 대하여 네 개 생겨나 있게 된다. 파일의 내용을 확인해 보아라.
이제 이 파일들을 컴파일하게 하자. 앞서 배운 \write18을 이용하여,
다시 강조하지만 이것이 실제로 효과를 가져서 외부 컴파일이 되게 하려면 --shell-escape 옵션이 반드시 필요하다. 파일이 네 개이므로,

```
\ExplSyntaxOn
\cs_new:Npn \compile_all:nn #1 #2
{
    \exp_args:No \file_if_exist:nF { \jobname-#1-#2.txt }
    {
        \sys_shell_now:x { xelatex~\jobname-#1.tex }
    }
}

\seq_mapthread_function:NNN \l_tmpa_seq \l_tmpb_seq \compile_all:nn
\ExplSyntaxOff
```

이렇게 하면 되겠다. 여기서는 \jobname이라는 매크로가 있기 때문에 이를 먼저 확장하여 shell로 보내어야 한다. 그래서 \sys_shell_now:x를 사용했다.
이제 각각의 tex이 컴파일되면서 예컨대 esg008-9pt-nine.txt라는 파일이 생겨나 있어야 한다. 이런 파일이 네 개 생기는 거다.
이 파일을 불러들이면 된다. 이 파일들은 한 줄씩 읽을 이유가 없고 전체를 다 읽어야 한 개의 clist를 설정하는 것이므로 다음과 같이 하여 한번에 읽어들여서 표를 만든다.

```
\protected\def\newlinehline{ \tabularnewline \hline }

\ExplSyntaxOn
\file_if_exist_input:n { \jobname-9pt-nine.txt }
\file_if_exist_input:n { \jobname-10pt-ten.txt }
\file_if_exist_input:n { \jobname-11pt-eleven.txt }
\file_if_exist_input:n { \jobname-12pt-twelve.txt }
```

```
\seq_set_from_clist:Nn \l_fontcmd_seq { miniscule, tiny, scriptsize, small,
 ↪  normalsize, large, Large, LARGE, huge, Huge, HUGE }

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

\seq_map_inline:Nn \l_fontcmd_seq
{
    \int_incr:N \l_tmpa_int
    \tl_put_right:Nn \l_tmpa_tl
    {
        \texttt{ \bs #1} \c_alignment_token
    }

    \clist_pop:NN \l_nine_clist \l_tmpb_tl
    \tl_put_right:Nx \l_tmpa_tl
    {
        \l_tmpb_tl &
    }

    \clist_pop:NN \l_ten_clist \l_tmpb_tl
    \tl_put_right:Nx \l_tmpa_tl
    {
        \l_tmpb_tl &
    }

    \clist_pop:NN \l_eleven_clist \l_tmpb_tl
    \tl_put_right:Nx \l_tmpa_tl
    {
        \l_tmpb_tl &
    }

    \clist_pop:NN \l_twelve_clist \l_tmpb_tl
    \tl_put_right:Nx \l_tmpa_tl
    {
        \l_tmpb_tl \newlinehline
    }
}

\begin{tabular}{r|r|r|r|r }
\hline
font~size~command & 9pt & 10pt & 11pt & 12pt \\ \hline
\l_tmpa_tl
\end{tabular}
\ExplSyntaxOff
```

font size command	9pt	10pt	11pt	12pt
\miniscule	5.0pt	5.0pt	6.0pt	7.0pt
\tiny	5.0pt	6.0pt	7.0pt	8.0pt
\scriptsize	6.0pt	7.0pt	8.0pt	9.0pt
\small	8.0pt	9.0pt	10.0pt	10.95pt
\normalsize	9.0pt	10.0pt	10.95pt	12.0pt
\large	10.0pt	10.95pt	12.0pt	14.4pt
\Large	10.95pt	12.0pt	14.4pt	17.28pt
\LARGE	12.0pt	14.4pt	17.28pt	20.74pt
\huge	14.4pt	17.28pt	20.74pt	24.88pt
\Huge	17.28pt	20.74pt	24.88pt	24.88pt
\HUGE	20.74pt	24.88pt	24.88pt	24.88pt

이 방법은 놀랍게도 원하는 결과를 얻게 한다. 그런데 실용적으로 생각해보면 컴파일 과정에서 외부 컴파일을

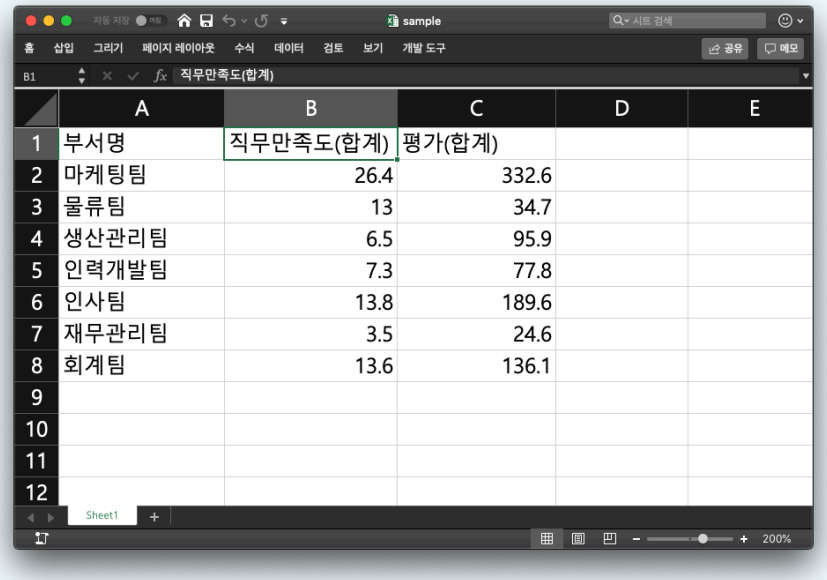
네 번이나 하는 것은 매우 지루하다. 여러 번 컴파일해야 하는 문서에서는 컴파일할 때마다 외부 컴파일도 반복된다. 이를 좀 줄이는 방법으로 `esg008-9pt-nine.txt`가 있으면 컴파일러를 부르는 명령을 실행하지 않는 방법이 있기는 하다. (위의 예에서 그렇게 했다. 여기서는 간단히 파일의 존재여부만 체크했지만 `\file_compare_timestamp:nTF`를 이용하여 파일의 변경 여부를 체크하는 것도 가능하다.)
그러나 일반적으로 말하자면 shell escape로 다른 프로그램도 아니고 `latex` 컴파일러 자체를 부르는 것은 좋다고 말하기 어렵다. 그보다는 컴파일 과정을 batch process로 만드는 것이 훨씬 효율적이다. `Makefile`을 이용하거나 `arara`를 이용하거나 컴파일 자체는 배치 컴파일로 돌리는 방법을 강구하자.
이 샘플은 하여간 이렇게 되기는 한다는 것을 보여주기 위한 예이다.

연습문제

응용 14. LaTeX 표준 클래스 `article` 단면(oneside) 단단(onecolumn) 문서에서 문서 용지 옵션이 `letterpaper`, `a4paper`, `a5paper`, `b5paper`일 때 각각 paperwidth, paperheight, textwidth가 어떻게 설정되어 있는지 조사하고 이를 한 개의 표로 나타내어 보아라. 표시 단위는 mm로 한다.

예제

다음 그림과 같은 Excel 파일(`sample.xlsx`)이 있다. 이 파일을 LaTeX 문서에 들여와서 tabular로 그리고, `\ShVal{B}{2}`라고 하면 B2 셀의 값인 26.4를 출력하는 명령 `\ShVal`를 작성하여라.



	A	B	C	D	E
1	부서명	직무만족도(합계)	평가(합계)		
2	마케팅팀	26.4	332.6		
3	물류팀	13	34.7		
4	생산관리팀	6.5	95.9		
5	인력개발팀	7.3	77.8		
6	인사팀	13.8	189.6		
7	재무관리팀	3.5	24.6		
8	회계팀	13.6	136.1		
9					
10					
11					
12					

33 csv 파일

스프레드시트 프로그램(Excel, LibreOffice Calc, etc.)은 고유의 포맷과 더불어 csv (comma-separated version)라는 텍스트 형식의 파일 포맷을 읽고 쓸 수 있다. csv는 plain text 파일이므로 이를 이용하여 Excel 데이터를 처리할 수 있다.
엑셀에서 "다른 이름으로 저장"을 선택하면 저장할 수 있는 형식 중에 CSV가 있다. 기본 열 분리자는 쉼표(,)이다. 또는 "탭으로 분리된 텍스트"를 선택하여 `*.txt`로 저장할 수 있는데 이것도 일종의 csv 포맷으로 본다. LibreOffice Calc는 csv 저장시에 분리자를 선택하는 옵션이 있을 것이다.
우리는 `xlsx2csv`라는 유틸리티를 사용하려 한다. 이것은 다음과 같이 하여 설치할 수 있다. `pip`은 버전에 맞는 명령을 주면 되고 python과 pip이 설치되어 있어야 한다.

```
pip install xlsx2csv
```

우분투 18.04에서

```
sudo apt install xlsx2csv
```

로 되었던 기억이 있다.
열 분리자를 무엇으로 하면 좋을까? csv라는 이름은 "쉼표"임을 강력히 내세우고 있지만 LaTeX 상황에서 쉼표는 생각을 좀 많이 해야 한다. 왜냐하면 텍스트 중에 쉼표가 들어 있다면 그것을 열 분리자와 구분하기 위해 모두 `{,}`로 묶어주는 번거로운 일을 감수해야 하기 때문이다.
그러나 텍스트 속의 쉼표 문제가 전혀 발생하지 않는다면 쉼표로 구분하는 것이 좋은데 각 행을 clist로 간단히 처리할 수 있을 것이기 때문이다.
열 분리자를 tab 문자(`\t`)로 하는 방법도 있다. 이호재 선생같은 경우 이 형식을 "tab separated version"이라고 TSV라고 부르기도 하는데 본질은 여전히 csv이므로 그렇게까지 구별해서 부를 이유가 있는지 모르겠고 아무튼

```
xlsx2csv sample.xlsx sample.csv
xlsx2csv -d tab sample.xlsx sample.csv
```

윗줄은 쉼표로 분리된 파일을, 아랫쪽은 tab으로 분리된 파일을 얻는다. 확장명은 둘 다 csv이다.
쉼표로 분리하여 sample.csv를 읽어들이면

```
\verbatiminput{sample.csv}
```

```
부서명,직무만족도(합계),평가(합계)
마케팅팀,26.4,332.6
물류팀,13,35
생산관리팀,6.5,95.9
인력개발팀,7.3,77.8
인사팀,13.8,189.6
재무관리팀,3.5,24.6
회계팀,13.6,136.1
```

33.1 표 그리기

한 줄씩 읽어서 표를 그리는 것은 간단할 것 같다. 한 번 해보자.

```
\protected\def\newlinehline { \tabularnewline \hline }
\ExplSyntaxOn
\ior_new:N \l_xls_ior
\ior_open:Nn \l_xls_ior { sample.csv }
\tl_clear:N \l_tmpb_tl

\ior_str_map_inline:Nn \l_xls_ior
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_replace_all:Nnn \l_tmpa_tl {, } { & }
    \regex_replace_all:nnN { $ } { \c{newlinehline } } \l_tmpa_tl

    \tl_put_right:Nx \l_tmpb_tl { \l_tmpa_tl }
}

\ior_close:N \l_xls_ior

\begin{tabular}{|c|c|c|}
\hline
\l_tmpb_tl
\end{tabular}
\ExplSyntaxOff
```

부서명	직무만족도(합계)	평가(합계)
마케팅팀	26.4	332.6
물류팀	13	35
생산관리팀	6.5	95.9
인력개발팀	7.3	77.8
인사팀	13.8	189.6
재무관리팀	3.5	24.6
회계팀	13.6	136.1

위의 지나치게 간단한 해법은 다음 조건이 충족되었기 때문에 문제를 일으키지 않는다. (1) 각 셀의 텍스트에 쉼표(,)가 없다. (2) 각 셀의 텍스트에 앰퍼선드(&)가 없다. (3) 각 셀의 텍스트에 LaTeX 명령이 없다.

`\ior_str_...`으로 읽어온 이유는 이런 류의 외부 파일에는 LaTeX이 불평하기 쉬운 문자들 (#, ^, _, % 등)이 포함되어 있을 가능성이 매우 높기 때문이다. 이런 문자들을 에러없이 처리하려면 string으로 읽어오는 것이 좋다. 단 string으로 읽어오면 TeX 명령이 포함되어 있을 때 이것이 실행되지 않는다.

33.2 csv의 셀 분리

B2 셀의 값을 얻으려 한다. 여기서 생각해보아야 할 것은
① 데어터 전체의 각 행을 하나씩 clist에 넣어두고 그 m번째 clist의 n번째 item을 출력하는 방법
② 파일을 m행까지 읽어가서 그 행의 n번째 item을 출력하는 방법
둘 가운데 어느 것이 효율적이겠느냐 하는 것이다. 우리의 지금 샘플과 같이 행이 몇 되지 않는 csv의 경우는 어떤 것이든 별 차이 없겠지만 행이 아주 많은 table이라면 그 모두를 clist에 넣는 것은 비효율적이다. 또 특정 셀의 값을 알아내는 명령이 얼마나 자주 호출되느냐도 중요한데 clist에 저장해두는 것은 이런 호출이 매우 자주 일어날 때라면 오히려 나을 수도 있을 것이다.
주어진 행까지 파일을 읽어가서 해당 행 하나만 얻는 방법으로 해보자.

```
\ExplSyntaxOn

\int_new:N \l_col_int
\int_new:N \l_row_int

\NewDocumentCommand \ShVal { m m }
{
    \int_set:Nn \l_col_int { \int_from_alph:n { #1 } }
    \int_set:Nn \l_row_int { #2 }

    \ior_open:Nn \l_xls_ior { sample.csv }

    \int_zero:N \l_tmpa_int

    \ior_str_map_inline:Nn \l_xls_ior
    {
        \int_incr:N \l_tmpa_int

        \int_compare:nT { \l_tmpa_int == \l_row_int }
        {
            \tl_set:Nn \l_mthline_tl { ##1 }
            \ior_map_break:
        }
    }

    \ior_close:N \l_xls_ior

    \seq_set_from_clist:NN \l_a_seq \l_mthline_tl

    \seq_item:Nn \l_a_seq { \l_col_int }
}
\ExplSyntaxOff

\ShVal{B}{4}
```

6.5

이번에는 각 행을 모두 각각의 seq에 넣어보겠다.

```
\ExplSyntaxOn

\NewDocumentCommand \ShVal { m m }
{
    \ior_open:Nn \l_xls_ior { sample.csv }

    \int_zero:N \l_row_int

    \ior_str_map_inline:Nn \l_xls_ior
    {
        \int_incr:N \l_row_int
        \seq_set_from_clist:cn { l_ \int_to_roman:n { \l_row_int } _ seq }
        { ##1 }
    }

    \ior_close:N \l_xls_ior

    \seq_if_exist:NT \l_i_seq { \int_set:Nn \l_col_int { \seq_count:N
     ↪  \l_i_seq } }

    \seq_item:cn { l_ \int_to_roman:n { #2 } _ seq }
    { \int_from_alph:n { #1 } }
    \c_space_token (
    \(
        \int_use:N \l_row_int \times \int_use:N \l_col_int
    \)
    )
}

\ExplSyntaxOff

\ShVal{C}{2}
```

332.6 (8 × 3)

위의 코드 중 재미있는 것은 `\ShVal`이 불리고 나면 `\l_col_int`와 `\l_row_int`가 이 table의 열과 행의 수를 보관하고 있다는 것이다. 출력의 괄호 안에 적어보았다. 이 int 변수를 사용해서 명령 인자가 행과 열의 범위를 벗어나는지 검사하여 에러 처리를 할 수 있다.
그리고 눈여겨 볼 것은 `\tl_set:Nn`과 같이 `\seq_set_...` 함수도 만약 해당 seq가 new 되어 있지 않으면 새로 만들고 설정해준다. 위와 같은 코드에서 `\seq_new:N`를 일일이 해야 한다면 그것도 귀찮은 일이었을 것이다.

33.3 csv와 LaTeX

LaTeX 패키지 가운데 csv를 다루는 것은 다음과 같은 것이 있다: csvsimple, datatool, pgfplotstable. 이 중 csv뿐만 아니라 제법 본격적인 데이터베이스 도구를 제공하려는 야심찬 패키지가 datatool이고 pgfplotstable은 csv를 table로 그려주는 데 특화되어 있으며 일반적인 의미에서 csv를 적절하게 다루는 데는 csvsimple이 좋다고 본다.
이미 테스트해본 대로 csv를 expl3로 다루는 것은 매우 쉽다. 굳이 패키지에 의존하지 않고도 간단한 작업은 얼마든지 해낼 수 있을 것이다.

연습문제

기본 15. 첨부하는 파일 `homework.xlsx`는 회원 주소록이다. 이 데이터로부터 가로 7cm인 박스 안에 우편번호, 주소, 이름을 인쇄하여 잘라서 봉투에 붙일 수 있게 A4 용지에 여러 개를 인쇄하여라. 절취선은 별도로 그리지 않아도 좋다.

25467
강원도 강릉시 강문동 *** 번지
가 각 간 귀하

예제

지난번에 작성한 `\DWalk` 명령이 있었다. 이것을 배포가능한 파일 묶음으로 만들어 보아라.

34 Packages

LaTeX을 처음 배울 때 class와 package를 알게 된다. class는 문서에서 단 하나만 지정할 수 있고 패키지는 여러 개를 use할 수 있다고 하였다.
자신이 작성한 expl3 명령을 다른 사람도 사용할 수 있게 하려면 이를 패키지로 배포하여야 한다. 굳이 배포하지 않더라도 반복 사용하는 코드들을 모아서 개인적 패키지로 만들어두는 사람들이 많다.
패키지라는 개념은 LaTeX 2_ε에서 처음 도입된 것이다. Leslie Lamport가 만든 LaTeX에는 클래스와 패키지의 구별이 없었다. 그래서

```
\documentstyle[a4,fancyheadings,psfig]{article}
```

이런 식으로 문서를 시작했는데 그러면 `article.sty`와 `a4.sty`, `fancyheadings.sty` 등을 차례로 로딩하는 방식이었다. 그러다보니 중복정의, 코드의 충돌 등 심각한 문제가 발생하기 시작했고 이를 해결하기 위해서 클래스와 패키지를 구분하게 한 것이다.
클래스에 대한 이야기는 다른 곳에서 하기로 하고 여기서는 패키지만 문제삼겠다.
Expl3 시대에 와서 패키지는 크게 두 종류로 구분된다. 하나는 전통적인 LaTeX 2_ε 패키지이고 다른 하나는 "Expl Package"라는 것이다. 본질만을 보자면 둘 사이에 큰 차이는 없지만 Expl Package는 LaTeX 2_ε 패키지가 가지지 못하는 몇 가지 속성을 더 가지고 있다. 우리는 Expl Package에 대해서만 다룰 것이고 LaTeX 2_ε 패키지 작성 방법에 대해서는 더 말하지 않을 생각이지만 아래 몇 가지를 지적하겠다.

34.1 LaTeX 2_ε 패키지에 대한 몇 가지

sty 파일 패키지 파일은 `.sty` 확장명을 갖는다. 이 파일을 문서에 불러올 때는 오직 preamble에만 올 수 있는

```
\usepackage[<options>]{<name>}
```

이라는 문장을 쓴다.
`\RequirePackage`라는 문장도 가끔 볼 수 있을텐데, 이것은 주로 패키지 제작자를 위한 명령으로서 `\usepackage`가 `\documentclass` 명령 이전에 올 수 없다는 제한을 없앤 것이다. 이 명령을 사용자가 사용하지 못하게 한 이유는 `\documentclass` 이전에 오는 코드들이 클래스 코드와 충돌할 때에 이를 해결할 능력을 문서작성자에게 요구할 수 없기 때문으로서, 일상적인 LaTeX 작업에서는 이 명령을 쓰지 않는다.
`\usepackage`는 해당 패키지를 불러들이면서 다음과 같은 일을 추가로 한다. (1) 패키지를 등록하여 이후 같은 패키지가 또 불릴 적에 이를 무시할 수 있게 한다. (2) 패키지를 불러오기 직전에 `\makeatletter`를 두고 패키지 불러오기가 끝난 시점에 `\makeatother`를 삽입한다.

패키지의 identity 예를 들어 `simpletest.sty`라는 파일이 있다고 하자. 이 파일의 제일 처음에 보통 다음과 같은 두 행이 있음을 볼 수 있다.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{simpletest}[2019/07/14 version 1]
```

첫 줄의 \NeedsTeXFormat{LaTeX2e}는 이것이 LaTeX 2$_\varepsilon$ 패키지임을 의미한다. LaTeX 2$_\varepsilon$가 아닐 때 에러 메시지를 출력한다. 지금 LaTeX 버전이 몇십년째 LaTeX 2$_\varepsilon$라서 사실 이 선언이 별 필요없는 때도 많고 귀찮아서 생략하는 사람들도 생겨나고 있지만 웬만하면 적어두는 것이 좋다.
\ProvidesPackage 명령이 중요하다. 이것은 LaTeX 시스템에게 이 패키지의 아이덴티티를 전달하는 역할을 한다. 이 가운데 중요한 것은 "패키지의 이름"과 "날짜" 두 가지이다. 여기서 선언된 "패키지 이름"이 동일하면 그 패키지를 중복 로딩하지 않는다. 한편 "특정 날짜 이전에 작성된 패키지 파일은 로딩하지 말라"는 식으로 명령을 줄 수 있는데 아주 특별한 경우에 쓰인다.
가끔 파일 자체의 이름과 패키지가 선언한 이름이 다를 때가 있다. 이 때는 LaTeX이 이 패키지를 로드할 때 뭔가 이상하다는 메시지를 콘솔에 뿌려준다. 동작에 큰 이상이 있는 것은 아니지만 에러메시지가 불쾌하고 중복 로딩에 착오가 생길 가능성도 있으므로 이 둘은 일치시켜 두어야 할 것이다.

패키지의 옵션 패키지의 옵션은 브래킷 안에 쉼표로 분리된 리스트로 부여한다. 어떤 옵션이 패키지에게 주어졌는지를 검사하는 패키지 내부 명령으로 \ProcessOptions 류의 명령이 사용된다. 옵션이 전달되었을 때 그 옵션에 따라 어떤 동작을 취할 것인가는 전적으로 패키지 내부의 문제이다.
\PassOptionsToPackage라는 명령은 이후에 \usepackage할 같은 이름의 패키지에게 미리 정해진 옵션을 넘기라는 명령이다.

dtx라는 파일 옛날에는 LaTeX 패키지가 전부 .dtx와 .ins 파일로 배포되었다. 이 방법은 아직도 쓰이지만 굳이 알아둘 필요가 크지는 않아서 이게 뭔가만 간단히 설명하고 지나간다. .dtx (Documented LaTeX) 포맷은 *code*와 *document*를 하나로 결합한 파일이다. cweb이 일반 프로그래밍 언어에 대해서 동작하는 "문학적 프로그래밍" 툴이라면 .dtx는 LaTeX의 문학적 프로그래밍이라고 할 수 있다. .dtx 문서로부터 *code* 부분만 추려내기 위해서 .ins라는 docstrip 파일을 제공한다. latex을 .ins에 적용하여 컴파일하면 *code*에 해당하는 부분이 파일(주로 .sty)로 풀려나오고 .dtx 자체에 대해서 latex을 적용하면 "문서"가 컴파일되어 나온다.
이 포맷에 관심이 있다면 Scott Pakin의 "An Introduction to writing .ins and .dtx files"라는 글(*TUGboat*19:2 (2008))과 docstrip 패키지 문서를 참고하라. 우리는 .sty를 .dtx로 묶는 것까지 다루지는 않겠다.

34.2 패키지를 만들기 전에

패키지를 꾸릴려면 적어도 다음과 같은 것이 준비되어 있어야 한다. 우리가 dummy라는 이름의 패키지를 만들기로 하고

(1) 패키지로 조성된 파일 자체. dummy.sty.

(2) 패키지 사용안내 문서. 적어도 문서의 소스와 pdf가 함께 제공되어야 한다. dummy-doc.tex, dummy-doc.pdf. texdoc이 이 문서를 쉽게 찾아야 하므로 패키지 자체의 이름과 일치시켜 두는 것이 좋고 대부분 -doc이 붙는 정도가 허용된다. 패키지 제작자의 입장에서 세상에서 가장 귀찮은 일이 문서를 만드는 것이다. 그러나 사용법 문서가 없는 패키지는 하등 아무짝에도 쓸모가 없다. 그건 그냥 쓰레기. 만약 문서가 특별한 그림 파일 따위를 이용하고 있다면 그런 것도 함께 묶어서 소스를 컴파일하여 동일한 문서를 얻을 수 있게 해야 한다. 예전에는 패키지 문서에 "구현"을 상세히 서술하였는데 (dtx 문서의 성격상 코드 해설이 위주가 되는 경향을 띨 수밖에 없었다) 이것은 개발자에게 매우 중요한 정보이다. 그러나 당장 필요한 것은 사용자에게 필요한 사용법 정보일 것이다. 이것만이라도 문서로 만들어두어야 한다.

(3) 저작권. LaTeX 패키지는 보통 LPPL이라는 copyright로 배포한다. 저작권 표시는 반드시 남기는 것이 좋다.

(4) 상세한 주석. 만약 패키지를 유지-보수할 의사가 있다면 자신이 작성한 코드에 일일이 세밀한 주석을 달아두어라. 이 코드를 왜 이 모양으로 짰는지 적어두지 않으면 수습불가능한 상황이 되고 결국 업데이트마저 불가능하게 될 것이다. 이 주석은 패키지 파일 안에 주석문으로 써넣어둔다.

34.3 Expl Package

Expl Package의 Identity Expl 패키지는 먼저 다음과 같이 시작한다. expl3 패키지를 먼저 로드하기 때문에 \NeedsTeXFormat{LaTeX2e} 같은 라인은 불필요하다.

```
\RequirePackage{expl3}
\ProvidesExplPackage
    {dummy}
    {2017/02/13}
    {0.001}
    {a dummy package for learning expl3 programming}
```

\ProvidesExplPackage 명령은 네 개의 인자를 갖는다. ① 패키지의 이름, ② 날짜, ③ 버전, ④ 패키지 설명.
이 선언이 불리게 되면 그 이후부터 해당 파일의 끝까지 expl syntax 상태가 된다. 즉 \ExplSyntaxOn과 \ExplSyntaxOff를 별도로 지정하지 않아도 된다.
(Expl Package가 아닌 LaTeX 2ε 패키지에서는 \RequirePackage{expl3,xparse}를 선언하고 나서 expl3 코드를 \ExplSyntaxOn과 \ExplSyntaxOff 사이에 넣어야 한다.)

Expl Package의 옵션 실제로 LaTeX 2ε 패키지 제작의 가장 어려운 점은 옵션을 처리하는 것이었다. 보통 간단한 단어로 주어지는 옵션이야 어떻게든 할 수 있지만 LaTeX 2ε가 발달하면서 점점 수요가 많아진 <*key*> = <*value*> 옵션을 처리하는 것이 제법 난관이었는데 그러다보니 keyval이니 xkeyval이니 하는 복잡하기만한 추가 패키지가 요구되는 그런 상황이었던 것이다.
expl3를 써야 하는 이유 중의 하나가 이런 <*key*> = <*value*> 옵션을 너무나 간단하게 처리할 수 있다는 것이다. l3keys2e라는 패키지는 expl3의 keys 데이터를 패키지의 옵션으로 사용할 수 있게 만들어준다. 다음 보기를 보자.

```
\RequirePackage{l3keys2e}

\keys_define:nn { dummymain }
{
    clear  .bool_set:N = \g_clear_bool,
    print  .tl_set:N = \g_print_tl
}

\ProcessKeysOptions { dummymain }
```

이것이 전부다. 이제

```
\usepackage[print={Please Help Me}]{dummy}
```

라고 하면 이 패키지 내부에서 \g_print_tl의 값이 주어진 문자열이 될 것이다.

34.4 패키지 작성 실전: esgdwalk 패키지

이름 짓기 패키지를 만들기로 하였다면 그 이름을 지어주어야 한다. 제작자의 입장에서야 짧고 간단하고 기억하기 쉬운 패키지 이름과 명령 이름을 짓고 싶겠지만 웬만한 단어들은 이미 다들 써먹었을 것이 틀림없어서 그게 그렇게 간단하지 않다. 특히 사용자 인터페이스 함수는 물론이고 내부 함수의 이름도 어디의 누군가가

이미 써버린 것이 확실할 정도로 간단한 이름으로 해두면 곤란하다. 내부 함수 이름은 특히 어디에서도 중복되지 않도록 작성해두는 것이 좋다. 우리가 인터페이스 함수 이름을 간단히 `\walk`이나 `\dwalk`이라고 이름짓기 꺼리는 이유는 거기에 있다. 최대한 양보한 것이 대문자를 섞어쓰는 `\DWalk`이었던 것이다. 예컨대 `\randomwalk` 이런 좋은 이름을 누군가 선점했을까 그렇지 않았을까? 혼자서 쓰는 패키지는 아무래도 문제 없겠으나 일단 "배포"하기로 했다면 이런 점을 잘 고려하여야 한다.
조금 다른 이야기지만 얼마 전까지 (또는 지금도) TEX으로 문서를 만드시는 분들은 자기 편하자고 예를 들면 `\def\v#1{\vec{#1}}` 이런 식으로 해놓고 쓰는 경우가 정말 정말 많았다. 그런 원고를 받아서 편집하는 사람의 입장에서 이것은 말할 수 없이 짜증나는 일인데, `\v`는 ě할 때 쓰이는 표준 LATEX 명령인 것이다. 이걸 `\def`해버리면 어쩌라는 말인가? 게다가 두 분 이상의 원고를 받아서 한 권의 책으로 묶는 상황에서 이것은 수많은 재정의 충돌을 일으킨다. LATEX 2_ε 이후로 사용자는 preamble에서 `\def` 대신 `\newcommand`를 쓰라고 그렇게 강조를 해도 잘 말을 듣지 않는 사람들이 많다. 핵심은 "남들이 이미 썼을 만큼 짧고 쉬운 명령은 사제로 만들어 쓰지 말라"는 것이다.
패키지의 이름은 더 까다로운데 이쪽은 너무 길어져도 곤란하기 때문이다. 물론 옛날같은 `8.3` 제한이 있는 것은 아니지만.
한때 제작자 이름을 앞에 붙이는 방식의 명명법이 유행한 적이 있는데 CTAN에서 한참 전부터 패키지 이름에 "자기 이름 첫글자를 붙이지 말라"고 하고 있다. 그것만 보아서는 도무지 어디에 쓰는 물건인지 알기 어렵게 하는 원인이었기 때문인 듯하다.
지금 제작해보려 하는 패키지의 이름은 `esgdwalk`으로 하려고 하는데 위의 기준에 맞추어보면 앞에 붙은 `esg`라는 표현은 범용 패키지로 배포하려 할 때는 좋은 명명법이 아니라고 하겠다.

시작 부분의 주석 보통 이 위치에는 이 파일의 이름과 저작권자, 저작권 및 기타 정보(간략한 사용법 등)를 주석문으로 기록해둔다. 다음과 같이 시작한다.

```
%
% file: `esgdwalk.sty`
%
% (c) 2019 Nova de Hi.
%
% A short sample package for Expl3 Study Group of KTUG
%
% ======= LPPL =======
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3 or later is part of all distributions of LaTeX
% version 2005/12/01 or later.
%
% This work has the LPPL maintenance status `maintained'.
%
% The Current Maintainer of this work is Nova de Hi.
%
```

여기서는 라이센스를 LPPL로 했는데, 원한다면 GPL이나 BSD Lisence나 Creative Commons License 등을 채택할 수 있다. 아니면 전적으로 사유(proprietary)로 선언하고 필요하면 돈을 내고 사라고 써놓을 수도 있겠지.

Expl Package 선언 다음과 같이 선언한다. expl3를 require하는 선언은 위의 주석문 바로 다음 행에 행을 띄지 말고 적어야 하는데 `\ProvidesExplPackage` 선언이 오기 전까지는 expl syntax가 아직 아니므로 여기를 띄어두면 "의도하지 않은 스페이스"가 들어갈 수 있기 때문이다. 대부분의 경우 별 문제 없으나 습관을 이렇게 들여두는 쪽을 추천한다.

```
\RequirePackage{expl3,xparse,l3keys2e}
\ProvidesExplPackage
    { esgdwalk }
    { 2019/08/03 }
    { 1.0 }
    { a sample expl3 package }
```

이 선언이 있은 이후부터 바로 \ExplSyntaxOn 되어 있다.

필수 패키지 이 패키지를 위하여 반드시 필요한 패키지를 로드한다.

```
\RequirePackage{tikz}
\RequirePackage{esgutil}
```

이 경우에 esgutil이라는 패키지가 배포판의 범용 패키지가 아니므로 이것을 어디에서 받을 수 있는지 설명하거나 아니면 (저작권의 문제가 없는 경우에) 함께 포함하여 배포하여야 한다.

옵션의 처리 옵션을 딱 하나만 주어보자. [color=blue]라는 식의 옵션을 받아들여서 선의 색상을 주도록 하겠다.

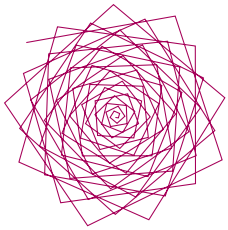
```
%%% package options
\keys_define:nn { dwalkpackage }
{
    color    .tl_set:N = \g_coloropt_tl
}

\ProcessKeysOptions { dwalkpackage }
```

이제 패키지 옵션은 \g_coloropt_tl로 넘어온다. 디폴트 색상을 주고 싶으면 \ProcessKeysOptions 명령 이전에 \keys_set:nn 해두면 될 것이다.

메인 코드 esg006에서 이미 만들어 본 \DWalk 코드를 그대로 가져오면 된다. 거기에서 또 keys 정의가 있는데 명령의 인자를 찾아내는 key를 module명을 dwalk으로 하였다. 이것이 패키지의 옵션을 받아오기 위한 keys의 module명과 달라야 할 것이다.
메인 코드는 첨부한 esgdwalk.sty의 소스 코드를 참고하여라. 이전에 했던 그대로이고 단지 색상 관련 코드만 추가하였다.
파일의 마지막에 \endinput을 두어서 파일의 끝임을 알려도 좋다.

문서 작성 이 패키지가 사용자에게 전달되었을 때 사용자는 패키지의 코드를 보는 것이 아니라 설명 문서를 원한다. 문서를 간단히라도 만들어보자. esgdwalk-doc.tex과 esgdwalk-doc.pdf가 문서에 해당한다.
여기까지 되면 다 된 것이다. 이 세 개의 파일 esgdwalk.sty, esgdwalk-doc.{tex,pdf}를 하나의 압축파일로 묶어서 배포한다.
\usepackage[color=red!30!violet]{esgdwalk}을 preamble에 두고 \DWalk 명령을 사용해보았다.

```
\DWalk{angle=82,forward=0.02+0.02,walk=100}
```



패키지의 배포 LaTeX 패키지는 CTAN을 통하여 배포한다. 자신이 작성한 패키지가 여러 사람에게 쓸모가 있고 버그가 없으며 사후 대응(버그 수정, 업데이트)에 자신이 있다면 CTAN에 업로드하여도 좋다.
한국적 상황에 맞는 패키지들은 KTUG Private Repository를 통하여 배포할 수도 있다.

연습문제

지금까지 자신이 작성한 함수와 명령 중에서 마음에 드는 것을 모아 자신의 이름을 붙여서 패키징하고 이를 KTUG Wiki의 “Expl3 Study Group” 페이지에 업로드하여라.