단어 첫 글자 장식

이엑스피엘쓰리 스타디 그룹

2019/02/20

요 약

입력되는 문자열(토큰열)을 단어 단위로 처리하는 몇 가지 방법을 보인다. expl3에서 재귀 함수를 어떻게 구성하고 활용하는지 설명한다.

차 례

1 문제

문제는 다음과 같은 것이다.

> 주어지는 문단의 각 단어 첫 글자를 특별히 장식하려 한다. 대문자로 만들거나, 색을 입히거나, 폰트를 다르게 하는 등의 조작을 할 수 있다. 인자는 "한 문단"으로 한정되며 \emph와 같은 일반적 명령은 단어 첫머리에 올 수 있다.

2 해결

2.1 한 단어의 첫 글자 조작

인자가 한 단어만 주어졌을 때 첫 글자만 조작하는 명령을 먼저 만들자. 이것을 \myfirstletter라고 하기로 한다.

```
\NewDocumentCommand \myfirstletter { m }
{
    \fbox{ \tl_head:n { #1 } }
```

```
    \tl_tail:n { #1 }
}
```

```
\myfirstletter{love}
```

l ove

이것으로 나쁘지는 않은데 문제는 \emph와 같은 명령이 단어와 함께 주어질 때일 것이다. 이를 위하여 다음처럼 수정한다.

```
\NewDocumentCommand \myfirstlettervar { m }
{
    \tl_set:No \l_tmpa_tl { \tl_head:n { #1 } }
    \exp_last_unbraced:Nx \token_if_macro:NTF \l_tmpa_tl
    {
        #1
    }
    {
        \fbox{ \tl_head:n { #1 } }
        \tl_tail:n { #1 }
    }
}
```

```
\myfirstlettervar{\emph{mytest}}
\\
\myfirstlettervar{mytest}
```

mytest
m ytest

\token_if_macro:NTF를 주의깊게 보면 되겠다. 이것은 입력문자열의 토큰이 macro인가 검사한다. 다만 주어진 단어의 첫 글자를 \l_tmpa_tl로 묶는 순간 이 매크로에 대하여 이 검사룰 수행하면 항상 true가 나올 수밖에 없으므로 이를 \exp_last_unbraced:Nx로 확장해주고 있다.

만약 첫 글자를 대문자나 소문자로 변환하려는 경우라면, KTUG 2015년 게시글 [1]에 예시되어 있는 \tl_to_uppercase는 더이상 동작하지 않는다. 이것은 초창기에 있었으나 현재는 사용이 금지된 expl3 명령이다. 그러므로 다음과 같이 하여야 한다. 위의 코드에서 \fbox했던 부분을 다음과 같이 수정한다.

```
\NewDocumentCommand \myfirstletteruppercase { m }
{
    \tl_mixed_case:n { #1 }
}
```

결과는 다음과 같이 나올 것이다.

```
    \myfirstletteruppercase{love}
```

Love

\tl_mixed_case:n은 첫 글자를 대문자로 만들어준다.

\tl_upper_case:n에 대응하는 \str_upper_case:n이 있다. 이들 사이의 구분은 "텍스트의 조판"에서 대/소문자를 식자하는 경우에는 '\tl_...'을 쓰고 그 외의 경우에는 '\str_...'를 쓰라는 것이라 한다. \tl_upper_case:n의 인자로 텍스트뿐 아니라 텍스트를 값으로 갖는 매크로가 오더라도 일단 모두 확장하여 텍스트로 만든 다음에 변환한다. 즉

```
\tl_set:Nn \l_tmpa_tl {world}
\tl_upper_case:n { \l_tmpa_tl }
```

이 조작의 결과는 "WORLD"이다. 반면 \str_upper_case:의 경우에 어떻게 되는지는 다음 예를 보라.

```
\str_upper_case:n { \l_tmpa_tl }⟹\L_TMPA_TL
\str_upper_case:f { \l_tmpa_tl }⟹WORLD
```

참고로 \str_ 타입은 그 뒤에 오는 모든 인자를 전부 catcode 12 (other)로 취급한다는 것을 기억해두자. 즉 매크로 이름이 들어와도 :n 인자지시형에서는 매크로를 실행하지 않고 매크로 문자 자체를 uppercase로 만들 것이다. 우리의 사례에서는 \tl_upper_case:n이 더 적절하리라고 본다.

2.2 문단을 단어 단위로 쪼개기: seq 접근

주어지는 한 문단을 단어 단위로 잘라서 위의 명령을 각 단어에 먹이면 되겠다. 연습을 위한 예문은 yihoze께서 즐겨 드는 마틴 루터 킹의 유명한 연설에서 한 문단을 가져다 쓰자.

이것은 인자로 들어오는 문단을 일단 seq에 space를 기준으로 split하여 넣어두고 이를 하나씩 처리(mapping)하겠다는 것이다. 다음 코드에서 \print_a_word:n은 앞서 논의한 바 한 단어를 처리하는 함수이다.

```
\NewDocumentCommand \testa { m }
{
    \seq_set_split:Nnn \l_tmpa_seq {~} { #1 }
    \seq_map_function:NN \l_tmpa_seq \print_a_word:n
}

\cs_new:Npn \print_a_word:n #1
{
    \tl_set:No \l_tmpa_tl { \tl_head:n { #1 } }
    \exp_last_unbraced:Nx \token_if_macro:NTF \l_tmpa_tl
    {
        #1
    }
    {
        \textcolor{blue}{\bfseries\itshape \tl_head:n { #1 } \/}
        \tl_tail:n { #1 }
        {}~
    }
}
```

결과는 예상대로이다.

```
\testa{With this faith, we will be able to hew out of the
   mountain of despair a stone of hope. With this faith, we
   will be able to transform the jangling discords of our
   nation into a beautiful symphony of brotherhood. With this
   faith, we will be able to work together, to pray together,
   to struggle together, to go to jail together, to stand up
   for freedom together, knowing that we will be free one day.}
```

***W**ith **t**his **f**aith, **w**e **w**ill **b**e **a**ble **t**o **h**ew **o**ut **o**f **t**he **m**ountain **o**f **d**espair **a** **s**tone **o**f **h**ope. **W**ith **t**his **f**aith, **w**e **w**ill **b**e **a**ble **t**o **t**ransform **t**he **j**angling **d**iscords **o**f **o**ur **n**ation **i**nto **a** **b**eautiful **s**ymphony **o**f **b**rotherhood. **W**ith **t**his **f**aith, **w**e **w**ill **b**e **a**ble **t**o **w**ork **t**ogether, **t**o **p**ray **t**ogether, **t**o **s**truggle **t**ogether, **t**o **g**o **t**o **j**ail **t**ogether, **t**o **s**tand **u**p **f**or **f**reedom **t**ogether, **k**nowing **t**hat **w**e **w**ill **b**e **f**ree **o**ne **d**ay.*

이 방법에서 주의해야 할 점은 한 단어를 처리하는 함수의 마지막에 '스페이스'를 반드시 하나 넣어주어야 한다는 것이다. space를 기준으로 잘라서 seq를 만들었기 때문에 각 아이템을 처리한 후에는 스페이스가 다시 복원되어야 한다. 당연히 맨 마지막 아이템에는 스페이스를 넣지 않는 루틴도 두는 것이 옳겠으나 여기에서는 그 부분을 생략하였다.

KTUG 게시물 [2]에는 긴 문단을 여러 단계의 seq로 split하여 조작하는 방법에 대한 설명이 나오는데 기본적으로 seq를 이용한다는 점에서 동일한 해법이다.

2.3 문단을 단어 단위로 쪼개기: 재귀 접근 (1)

expl3의 *quark*-자료형은 재귀적 처리를 위한 함수를 제공한다. 이를 간단히 스케치하면 다음과 같다.

(1) 재귀호출 함수(여기서는 `\my_fn:n`)를 앞세우고 범위표시(중괄호) 없이 재귀적으로 처리할 토큰열을 늘어놓는다. 이렇게 하면 `\my_fn:n`은 일단 맨 처음 토큰 하나만을 취하여 어떤 일을 하게 될 것이다.

(2) 재귀적으로 처리할 토큰열이 끝나는 곳에 다음 두 개의 표지를 붙인다. `\q_recursion_tail`, `\q_recursion_stop`.

(3) `\my_fn:n`의 정의 맨처음에 `\quark_if_recursion_tail_stop:n`을 두어서 재귀처리를 중단하는 지점을 표시한다.

(4) `\my_fn:n`은 자기 자신을 재귀호출하게 정의한다.

이 규칙에 따라 간단한 시험을 하여 보면,

```
\NewDocumentCommand \testone { 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 }
    \fbox{#1}
    \test_fn:n
```

```
}
```

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

T h i s i s a t e s t

재귀적 함수를 구성하는 데 있어 가장 중요한 것은 그 중단지점을 명확하게 하는 것이다. expl3가 제공하는 이 방법을 사용하면 중단조건에 대하여 큰 고민없이 재귀적으로 함수를 쓸 수 있게 하므로 매우 편리하다.

우리 과제로 다시 돌아와서 해결해야 할 문제를 생각해보면 두 가지가 있다. 하나는 인자로 주어진 텍스트를 단어 단위로 분리하는 것이고 다른 하나는 분리된 단어를 처리하는 것이다.

이를 위해 다음과 같은 방법을 사용할 것이다.

(1) 토큰 하나를 취한다.

(2) 하나의 단어를 임시로 모아두는 *tl*에 들어온 토큰을 넣는다.

(3) 다음 토큰이 space인지 검사한다.

(4) 만약 다음 토큰이 space이면, 현재까지 모인 *tl*을 현재 위치에 식자하고 해당 *tl*을 비운 다음 space 하나를 출력한다.

(5) 그 다음 토큰으로 이동한다.

```
\NewDocumentCommand \testw { m }
{
    \tl_clear:N \l_tmpa_tl
    \testw_fn:n #1 \q_recursion_tail \q_recursion_stop
}

\cs_new:Npn \testw_fn:n #1
{
    \quark_if_recursion_tail_stop:n { #1 }
    \tl_put_right:Nn \l_tmpa_tl { #1 }
    \peek_catcode:NTF \c_space_token
    {
        \fbox{\l_tmpa_tl}
        \tl_clear:N \l_tmpa_tl
        {}~
        \testw_fn:n
    }
    {
        \testw_fn:n
    }
}
```

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

This is just a

일단 원하는 결과가 나오는 것 같기는 하다. 그런데 마지막 단어가 사라져 있다. 이것은 당연한 것이, \q_recursion_tail과 \q_recursion_stop을 만나면 재귀를 중단하라고 하였기 때문이다. 이것이 의미하는 바는, 여기서 재귀호출을 중단할 적에 그냥 바로 중단하지 말고 뭔가 일을 하고 끝내야 한다는 의미이다. 이를 위하여 해당 부분을 다음과 같이 수정한다.

```
\NewDocumentCommand \testww { m }
{
    \tl_clear:N \l_tmpa_tl
    \testww_fn:n #1 \q_recursion_tail \q_recursion_stop
}

\cs_new:Npn \testww_fn:n #1
{
    \quark_if_recursion_tail_stop_do:nn { #1 }
    {
        \fbox{\l_tmpa_tl}
    }

    \tl_put_right:Nn \l_tmpa_tl { #1 }

    \peek_catcode:NTF \c_space_token
    {
        \fbox{\l_tmpa_tl}
        \tl_clear:N \l_tmpa_tl
        {}~
        \testww_fn:n
    }
    {
        \testww_fn:n
    }
}
```

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

This is just a test

\quark_if_recursion_tail_stop_do:nn은 \q_recursion_tail에 의하여 재귀호출이 종료되었을 때 그 직후에 할 일을 지정해줄 수 있다. 여기서는 마지막 한 단어를 출력하는 것이다.

\fbox 대신 앞서 정의한 \print_a_word:n을 이용하여 예문을 출력해보자. \testv 명령은 \testv_fn:n을 재귀함수로 부르는데 \fbox 대신

```
\print_a_word:V \l_tmpa_tl
```

를 사용한 것이다. \print_a_word:n을 이미 정의해두었으므로 그 변형인 :V는

```
\cs_generate_variant:Nn \print_a_word:n { V }
```

와 같이 하여 활용할 수 있다.

```
\NewDocumentCommand \testv { m }
{
    \tl_clear:N \l_tmpa_tl
    \testv_fn:n #1 \q_recursion_tail \q_recursion_stop
}

\cs_generate_variant:Nn \print_a_word:n { V }

\cs_new:Npn \testv_fn:n #1
{
    \quark_if_recursion_tail_stop_do:nn { #1 }
    {
        \print_a_word:V \l_tmpa_tl
    }

    \tl_put_right:Nn \l_tmpa_tl { #1 }

    \peek_catcode:NTF \c_space_token
    {
        \print_a_word:V \l_tmpa_tl
        \tl_clear:N \l_tmpa_tl
        \testv_fn:n
    }
    {
        \testv_fn:n
    }
}
```

```
\testv{With this faith, we will be able to hew out of the
   mountain of despair a stone of hope. With this faith, we
   will be able to transform the jangling discords of our
   nation into a beautiful symphony of brotherhood. With this
   faith, we will be able to work together, to pray together,
   to struggle together, to go to jail together, to stand up
   for freedom together, knowing that we will be free one day.}
```

*W*ith *t*his *f*aith, *w*e *w*ill *b*e *a*ble *t*o *h*ew *o*ut *o*f *t*he *m*ountain *o*f *d*espair *a* *s*tone *o*f *h*ope. *W*ith *t*his *f*aith, *w*e *w*ill *b*e *a*ble *t*o *t*ransform *t*he *j*angling *d*iscords *o*f *o*ur *n*ation *i*nto *a* *b*eautiful *s*ymphony *o*f *b*rotherhood. *W*ith *t*his *f*aith, *w*e *w*ill *b*e *a*ble *t*o *w*ork *t*ogether, *t*o *p*ray *t*ogether, *t*o *s*truggle *t*ogether, *t*o *g*o *t*o *j*ail *t*ogether, *t*o *s*tand *u*p *f*or *f*reedom *t*ogether, *k*nowing *t*hat *w*e *w*ill *b*e *f*ree *o*ne *d*ay.

2.4 문단을 단어 단위로 쪼개기: 재귀 접근 (2)

expl3의 l3quark이 제공하는 편리한 재귀 관련 함수를 이용하지 않아도 함수의 재귀적 정의를 활용하는 것은 얼마든지 가능하다. 단 그 "종료조건"을 잘 관리해야 한다.

이번에는 다음과 같은 형태의 명령을 만들어보고 싶다. 인자를 주는 것이 아니라 그냥 명령을 앞세우고 중괄호 없이 텍스트를 두고자 하는 것이다. 여기서는 편의상 종료 위치에 |를 두어서 끝임을 알리도록 하려 한다. 그러면 명령은 대략 다음과 같은 꼴이 될 것이다.

```
\mytest With this faith,| we well be able to hew out of ...
```

\mytest 명령은 With this faith,까지만 적용된다.

기본적인 아이디어는 앞서와 동일하다. 바로 코드를 보자. 종료 조건의 처리에 유의해서 보면 되겠다.

```
\NewDocumentCommand \testc {}
{
    \tl_clear:N \l_tmpa_tl
    \testc_fn:n
}

\cs_new:Npn \testc_fn:n #1
{
    \str_if_eq:nnTF {|} { #1 }
    {
        \fbox{\l_tmpa_tl}
    }
    {
        \tl_put_right:Nn \l_tmpa_tl { #1 }

        \peek_catcode:NTF \c_space_token
        {
            \fbox{\l_tmpa_tl}
            \tl_clear:N \l_tmpa_tl
            {}~
            \testc_fn:n
        }
        {
            \testc_fn:n
        }
    }
}
```

```
\testc With this faith,| we well
   be able to \ldots
```

With this faith, we well be able to …

\fbox한 부분을 \print_a_word:n로 바꾸면 되므로 더이상의 설명은 생략한다.

2.5 문단을 단어 단위로 쪼개기: regex 접근

expl3가 제공하는 가장 강력한 도구 중의 하나가 l3regex일 것이다. 이를 이용하는 법에 대해 생각해본다. 이를테면 다음과 같이 되어 있는 입력 문자열이 있다고 하자.

```
With<sp>this<sp>faith,<sp>we<sp>will
```

여기서 <sp>란 스페이스를 가리킨다. 만약 이 <sp> 위치에 "한 단어를 식자하는 명령"이 온다면 어떻게 될까?

```
\spfn With \spfn this \spfn faith, \spfn we \spfn will
```

사실 이것이야말로 우리가 지금 하려는 일이다. 즉, \spfn이 하는 일이 \print_a_word:n와 같다면, 그리고 그 다음 한 단어가 이 명령의 인자로 주어질 수만 있다면 원하는 것을 이루는 셈이다. regex로 뭔가를 해볼 수 있을 듯하다.

그렇다면 \spfn은 어떻게 정의되어야 하는 걸까? 다음을 보자.

```
\NewDocumentCommand \spfn {}
{
    \tl_clear:N \l_tmpa_tl
    \sp_fn:n
}

\cs_new:Npn \sp_fn:n #1
{
    \str_case:nnF { #1 }
    {
        { | } { \fbox{\l_tmpa_tl }~ }
        { \spfn } {
            \fbox{\l_tmpa_tl}~
            \tl_clear:N \l_tmpa_tl
            \sp_fn:n
        }
    }
    {
        \tl_put_right:Nn \l_tmpa_tl { #1 }
        \sp_fn:n
    }
}
```

```
\spfn With \spfn this \spfn
faith,| we will
```

With this faith, we will

이 함수는 입력 토큰열에서 한 개의 토큰을 취한다. 그것이 \spfn이라는 이름의 토큰이면 현재까지 구성된 \l_tmpa_tl을 출력하고 비운 다음 다음 토큰으로 이동한다. 만약 들어온 토큰이 |이면 \l_tmpa_tl을 출력하는 일만을 하고 다음 토큰으로 이동하는 일을 하지 않는다. 즉 그 위치에서 재귀 호출이 종료되는 것이다. 이 두 가지가 아니라면 현재 토큰을 \l_tmpa_tl이 넣고 다음으로 이동한다.

준비가 갖추어졌다. 이제 해야 할 일은 실제로 입력되는 인자에서 'space'를 모두 '\spfn'으로 교체해 주는 일이다.

```
\NewDocumentCommand \testr { m }
{
    \tl_set:Nn \l_tmpb_tl { #1 }
    \regex_replace_all:nnN { \h } { \c{spfn} } \l_tmpb_tl
    \exp_last_unbraced:No \spfn \l_tmpb_tl |
}
```

\regex_... 함수에서 \h는 스페이스와 탭문자에 해당한다. 이 두 종류의 문자를 모두 \spfn으로 바꾸라는 것이다. 당연한 것이지만 마지막에 종지부호 |를 붙였고 \l_tmpb_tl을 중괄호 없이 확장하기 위해 \exp_last_unbraced:No를 썼다.

```
\testr{With this faith, we will be able to hew out of the
   mountain of despair a stone of hope. With this faith, we
   will be able to transform the jangling discords of our
   nation into a beautiful symphony of brotherhood. With this
   faith, we will be able to work together, to pray together,
   to struggle together, to go to jail together, to stand up
   for freedom together, knowing that we will be free one day.}
```

***W**ith **t**his **f**aith, **w**e **w**ill **b**e **a**ble **t**o **h**ew **o**ut **o**f **t**he **m**ountain **o**f **d**espair **a** **s**tone **o**f **h**ope. **W**ith **t**his **f**aith, **w**e **w**ill **b**e **a**ble **t**o **t**ransform **t**he **j**angling **d**iscords **o**f **o**ur **n**ation **i**nto **a** **b**eautiful **s**ymphony **o**f **b**rotherhood. **W**ith **t**his **f**aith, **w**e **w**ill **b**e **a**ble **t**o **w**ork **t**ogether, **t**o **p**ray **t**ogether, **t**o **s**truggle **t**ogether, **t**o **g**o **t**o **j**ail **t**ogether, **t**o **s**tand **u**p **f**or **f**reedom **t**ogether, **k**nowing **t**hat **w**e **w**ill **b**e **f**ree **o**ne **d**ay.*

3 생각해볼 것

TEX에서 재귀, 즉 꼬리재귀란 실제 식자공의 조판 과정을 흉내낸 것이라고 생각하면 매우 재미있다. 하나의 글자를 'type-set'하고, 잘 되었으면 다음 글자를 또 'type-set'한다. 요컨대 동일한 과정을 입력되는 각각의 문자에 대하여 차례로 적용해가는 것이 조판과정이라면 TEX의 재귀 호출 역시 동일한 과정을 입력되는 토큰에 하나씩 차례로 적용해가는 것이다.

expl3에서 재귀 방식으로 함수를 작성하는 것은 매우 쉽다. 오류를 피하기 위한 장치도 잘 마련되어 있다. 아주 약간만 주의를 기울이면 손쉽게 이 테크닉을 자유자재로 활용할 수 있을 것이다.

또한 기존에 '재귀'로 처리하던 많은 절차를 미리-정의된 함수들로 할 수 있게 해두었다. expl3에서 가장 요긴한 것을 하나 고르라면 역시 \<type>_map_inline:과 \<type>_map_function:일 터인데 이것이야말로 사실 '재귀'를 구현한 것이라고 보아도 무방하다.

본문의 몇 가지 해결책 가운데 아마도 seq 방법이 가장 이해하고 구현하기 쉽다고 느낄 것인데, 그것이 사실이다. 그것이면 충분하지 않을까?

마지막으로 지적해둘 것은, 위에 소개한 몇 가지 방법이 모두 공통적으로 hyphenation의 단어 잘림이 일어나지 않는다. 이것은 다른 문제이므로 여기서는 더 다루지 않았다.

참고 글

[1] yihoze, negatropy, and NDH, "단어의 첫 글자만 다르게," http://www.ktug.org/xe/index.php?mid=KTUG_QnA_board&m=0&document_srl=214192.

[2] yihoze, Progress, and noname, "단어에 색을 입힐 때," http://www.ktug.org/xe/index.php?document_srl=234545&mid=KTUG_QnA_board.

[3] yihoze, "자모에 색 입히기," http://www.ktug.org/xe/index.php?document_srl=234587&mid=KTUG_open_board.