

srcandtext

Nova de Hi

Feb 2025 v0.9

요약

srcandtext 패키지에 관하여 설명한다. expl3 프로그래밍에 관련된 글을 쓸 적에 예시 코드를 제시하고 이를 실행하기 위하여 두 번 소스를 적어야 하는 불편을 없애고 expl3를 “문학적 프로그래밍” 방식으로 작성할 수 있게 만들었다.

차례

1	목적과 배경	1
2	기본적인 동작	2
2.1	sntsave 환경과 \loadsnttext 명령	2
2.2	sntsave 환경의 expl	5
2.3	별표붙은 sntsave* 환경	5
3	클래스와 스타일	5
3.1	클래스 옵션	5
3.2	패키지 옵션	6
3.3	tcolorbox 옵션과 example 환경	7
3.4	자잘한 기능들	7
4	예문	8

1 목적과 배경

이 클래스는 짧은 예시 코드를 작성하고 이를 설명하고 그 실행 결과를 보여주는 문서를 작성하기 위하여 만들었다. expl3 관련 글을 몇 개 쓰면서 간단한 예제를 만들어 보이려 할 적에 다음과 같은 절차를 거쳐야 했다.

- (1) 코드를 작성한다.
- (2) 이 코드를 *<verbatim>* 형태로 식자한다.
- (3) 같은 코드를 실행한다.
- (4) 예제를 실행한다.

tcolorbox와 같은 훌륭한 패키지의 도움을 얻어서 소스와 그 실행 결과를 보이는 것은 어려운 일이 아니다. 예를 들면

<pre>\ExplSyntaxOn \dim_eval:n { \textheight - \textwidth } \ExplSyntaxOff</pre>	533.68134pt
--	-------------

이러한 일은 간단하게 된다.

그러나 예컨대 내가 다음과 같은 `expl3` 함수를 만들었다고 가정하자.

```
\justtest{Hello, world}
```

```
Hello, world (11)
```

이 함수가 하는 일은 입력된 문자열을 `\fbox`로 출력하고 문자열의 길이(스페이스 제외)를 괄호 안에 보이는 것이다. 이에 관한 설명 문서를 만들려면, 먼저 이 함수 자체를 코드로 문서에 넣어야 하고, 그 다음에 이것을 `verbatim`으로 보이고 그 출력을 제시해야 한다. 즉, 적어도 코드를 두 번은 복사해서 써넣어야 하는 것이다. 다음은 이 함수를 작성한 코드를 설명하기 위하여 작성해야 하는 소스의 예시이다.

```
\ExplSyntaxOn
\cs_new:Npn \justtest_fn:n #1
{
  \fbox { #1 } ~ ( \tl_count:n { #1 } )
}
\cs_set_eq:NN \justtest \justtest_fn:n
\ExplSyntaxOff

\begin{verbatim}
\ExplSyntaxOn
\cs_new:Npn \justtest_fn:n #1
{
  \fbox { #1 } ~ ( \tl_count:n { #1 } )
}
\cs_set_eq:NN \justtest \justtest_fn:n
\ExplSyntaxOff
\end{verbatim}

\justtest{Hello, world}
```

문서를 작성하다 보면 함수를 수정해야 할 때가 있는데 이 때는 괴롭다. 적어도 두 곳을 수정해야 하기 때문이다. 이 귀찮은 복사해서 붙이기를 좀 어떻게 해보기 위해서 이 클래스를 작성하게 되었다. 이 클래스의 방법을 쓰면 다음과 같이 코딩할 수 있다.

```
\autoexplon
\begin{sntsave}
%<Code>
\cs_new:Npn \justtest_fn:n #1
{
  \fbox { #1 } ~ ( \tl_count:n { #1 } )
}
\cs_set_eq:NN \justtest \justtest_fn:n
%</Code>
\end{sntsave}
\loadsnttext*
```

이 클래스는 스타일 파일을 문서와 코드로 제작하는 `dtx`와 유사한 아이디어로 만들었지만 `.sty` 파일에 대하여 쓸 수 있는 것은 아니다.

이 클래스의 아이디어는 tcolorbox가 제공하는 `verbatim`으로 한 번 보이고 코드로서 한 번 실행하는 기능에 착안한 바가 크다. 또한 동일한 소스를 코드로서 한 번 읽고 텍스트로서 한 번 읽게 한다는 것으로써 ‘문학적 프로그래밍’과 비슷하게 코딩과 그 해설을 동시에 작성할 수 있게 되었다.

2 기본적인 동작

2.1 sntsave 환경과 \loadsnttext 명령

다음과 같은 내용을 `<sntsave>` 환경 내에 넣었다.

```
%<Code>
\ExplSyntaxOn
%</Code>
%% \numpar 주어진 수까지 합을 구한다.
%%
%% 합을 저장할 장소 \stm{l_sum_int}를 준비한다.
%<Code>
\int_new:N \l_sum_int
%</Code>
%% \numpar 명령의 이름을 \stm{mysum}이라 하자.
%% 일단 \stm{l_sum_int}를 비운다.
%<Code>
\NewDocumentCommand \mysum { m }
{
  \int_zero:N \l_sum_int
%</Code>
%% \wi{expl3의 for loop}는 \stm{int_step_...} 인터페이스 함수로
%% 쓸 수 있다.
%<Code>
  \int_step_inline:nn { #1 }
  {
    \int_add:Nn \l_sum_int { ##1 }
  }
%</Code>
%% 여기서 |##1|과 같이 |#|이 두 번 쓰인 것이 중요하다.
%% 이것은 \stm{mysum}이라는 함수를 정의하는 코드이다. 따라서
%% 이 범위 내에서는 |#1|이 언제나 |\mysum|의 인자로 넘어온
%% 것을 가리키게 된다.
%% 반면 \stm{int_step_inline:} 루프 안에서의 |#1|은 for 루프의
%% 인덱스 카운터이다. |#1|과는 상관없는 것이다. 그래서 |#|을 하나
%% 더 붙여 |##1|이라고 한 것.
%%
%% 이제 합을 출력하고 종료한다.
%<Code>
  \int_use:N \l_sum_int
}

\ExplSyntaxOff
%</Code>
```

규칙은 매우 단순하다.

- (1) 두 개의 comment 기호 다음에 써넣는 것은 “코드로서 실행될 때는 주석이고 텍스트로서 실행될 때는 본문”이 되는 것이다.
- (2) 코드는 `%<Code>`와 `%</Code>` 범위 안에 둔다. 이것은 “코드로서 실행될 때는 코드이고 텍스트로서 실행될 때는 verbatim”이다.

진짜 주석문(comment)를 달고 싶다면 %를 세 개 이상 쓰면 된다. 왜냐하면 행 첫머리가 두 개의 %로 시작하는 행만을 텍스트로 복원하기 때문이다.

이제 이것은 텍스트로서 불러들일 수 있다. 이 때는

```
\loadsnttext
```

명령을 쓴다. 직접 결과를 보자.

```
\ExplSyntaxOn
```

1. 주어진 수까지 합을 구한다.

합을 저장할 장소 `\l_sum_int`를 준비한다.

```
\int_new:N \l_sum_int
```

2. 명령의 이름을 `\mysum`이라 하자. 일단 `\l_sum_int`를 비운다.

```
\NewDocumentCommand \mysum { m }
{
  \int_zero:N \l_sum_int
}
```

expl3의 for loop는 `\int_step_...` 인터페이스 함수로 쓸 수 있다.

```
\int_step_inline:nn { #1 }
{
  \int_add:Nn \l_sum_int { ##1 }
}
```

여기서 `##1`과 같이 `#`이 두 번 쓰인 것이 중요하다. 이것은 `\mysum`이라는 함수를 정의하는 코드이다. 따라서 이 범위 내에서는 `#1`이 언제나 `\mysum`의 인자로 넘어온 것을 가리키게 된다. 반면 `\int_step_inline:` 루프 안에서의 `#1`은 for 루프의 인덱스 카운터이다. `#1`과는 상관없는 것이다. 그래서 `#`을 하나 더 붙여 `##1`이라고 한 것.

이제 합을 출력하고 종료한다.

```
\int_use:N \l_sum_int
}

\ExplSyntaxOff
```

만약 이 명령에 별표를 붙이면,

```
\loadsnttext*
```

이 때는 텍스트로서 한 번 불러오고 코드로서 한 번 불러온다. 만약 이 코드에 `\input` 시의 에러가 발생한다면 어딘가 오류가 있는 것이다. 예컨대 `\ExplSyntaxOn`을 하지 않았단가.

아무튼 이와 같이 “코드로서” 불러들이는 데 성공하였다면 여기서 정의된 함수를 실제로 실행할 수 있다.

```
\mysum{10}
```

55

만약에 `<sntsave>` 범위의 코드 안에 “조판 가능한 material”이 포함되어 있다면, 이를테면 위의 `\mysum{10}`과 같은 식자할 수 있는 코드가 포함되어 있다면, “코드로서 불러올 때”에 이 부분이 실행된다. 우리의 예에서는 `<sntsave>`한 범위에는 오직 함수 정의만이 있기 때문에 이를 코드로서 불러들였을 때 아무 것도 식자되지 않을 뿐이다.

“텍스트로” 식자하지 않으면서 코드만을 불러들이려면 다음 명령을 쓴다:

```
\loadsntcode
```

`<sntsave>` 환경은 반복해서 쓸 수 있다. 그러나 `\loadsntcode`와 `\loadsnttext` 명령이 불러오는 것은 가장 마지막으로 저장된 `<sntsave>`의 내용이다.

2.2 sntsave 환경의 expl

원래 이 클래스가 expl3 교재를 저술하는 과정에서 나온 것이어서 예제가 거의 expl3인 상황에서 제작되었다. 그러다보니 매번 예제마다 `\ExplSyntaxOn`와 `\ExplSyntaxOff`를 지정하는 것이 귀찮은 것이었다.

이것을 자동으로 붙여주도록 할 수 있다. 클래스 옵션으로 expl을 선언하면 모든 `<sntsave>`와 `<sntsave*>` 환경에 대하여, 시작할 때에 `\ExplSyntaxOn`을, 끝날 때 `\ExplSyntaxOff`를 자동으로 추가한다. 이것은 “코드”에만 추가되는 것으로 “텍스트”로서 불러올 때에는 식자되지 아니한다.

자동으로 붙이는 동작을 바꾸려면 언제라도

- `\autoexplon`
- `\autoexploff`

를 선언한다. 이름 그대로 자동으로 expl syntax를 설정하는 동작을 켜거나 끄는 것이다.

2 페이지의 `boxedverbatim` 안에 들어 있는 예제는 `\autoexploff`한 상태에서 `\ExplSyntaxOn`을 직접 써 넣은 것이다. 반면 10 페이지의 `boxedverbatim` 입력 예제는 `\autoexplon` 상태에서 입력한 것으로 입력된 부분에는 `\ExplSyntaxOn` 등이 들어 있지 않으나 코드로서 불러들이면 자동으로 붙어 있는 예를 보여준다. 그렇기 때문에 이를 식자하는 명령 `\loadsnttext`가 불릴 때 전자는 `\ExplSyntax` 명령이 텍스트로 나타나지만 후자는 그렇지 아니한 것을 볼 수 있다.

2.3 별표붙은 sntsave* 환경

만약 문서 전체에 걸쳐서 설명하고자 하는 코드를 preamble에 두는 경우라면 `<sntsave*>` 환경을 preamble에서 쓸 수 있다. 이 환경은 문서 본문에서는 의미가 없고 preamble에서만 동작하는데 하는 일은 기본적으로 preamble에서 `<sntsave*>`로 정의된 코드를 한 번 로드해주는 것이다. 즉 `<sntsave*>`에 정의된 코드들이 마치

스타일을 불러들이는 것처럼 일단 한 번 읽는 상태가 된다. 따라서 preamble에서 이 환경을 사용하였다면 환경 내에서 정의된 명령과 함수를 문서 본문에서 쓸 수 있다.

단, 새로이 `<sntsave>`를 쓰지 않은 상태로 `\input`이나 `\loadsntcode`로 소스 코드를 다시 불러들이려 한다면 함수들이 중복 정의되었다고 불평할 것이다. 그러나 `\loadsnttext`로 preamble에서 정의한 함수를 해설과 코드를 식자할 수 있다. (`<sntsave*>`로 저장된 코드를 본문에서 식자할 때는 `\loadsnttext*`와 같이 하는 경우 코드를 두 번 읽게 되므로 에러가 발생할 수 있다. 별표는 둘 중의 하나만 쓰도록 한다.)

3 클래스와 스타일

이 패키지는 `srcandtext.cls`라는 클래스와 `stcandtext.sty`라는 스타일로 이루어져 있다. 스타일은 다른 클래스의 문서에서 독립적으로 사용하게 하기 위하여 제작한 것이다. version 0.4부터 원칙적으로 일반 클래스를 지원한다. 적어도 표준 \LaTeX 클래스는 테스트를 거쳤으나 그밖의 다른 클래스/스타일과의 충돌(재정의 오류)이 없다고 보증하지는 못한다. 이 패키지는 모든 소스 코드 리스팅을 `tcolorbox`에 의하고 있으므로 다른 소스 코드 리스팅 패키지를 되도록 쓰지 않는 것이 충돌을 줄이는 방법임을 적어둔다.

3.1 클래스 옵션

이 클래스는 다음과 같이 로드한다.

```
\documentclass[<options>]{srcandtext}
```

옵션으로 줄 수 있는 것은 다음과 같다.

paper `papersize`를 지정한다. default: `a4paper`.

margin `fapapersize`의 좌우상하 여백 크기를 지정할 수 있다. `margin={1in,*,2in,*}`와 같은 방법으로 지시할 수 있다. default: `{25mm,*,30mm,*}`.

parindent `\parindent` 길이를 지정한다. default: `0pt`.

parskip `\parskip` 값을 지정한다. default: `5pt plus 2pt minus 1pt`.

extraopt 기타 `oblivoir`에 먹일 옵션을 추가하려면 여기에 적어넣으면 된다. 예를 들면 `extraopt={11pt}`. default: `[none]`.

math `amsmath`를 로드하고 `ob-mathleading`을 불러온다. `ob-mathleading` 패키지는 \TeX Live로 배포되는 것이 아니라 KTUG 사설저장소로부터 설치할 수 있다. default: `false`.

bvlskip `verbatim` 부분의 행간 간격을 조절하는 값으로 쓰는 `baselinestretch` 값이다. `tcolorbox` 옵션이 주어졌을 때만 작동하며 `<boxedverbatim>`으로 식자할 적에는 효력이 없다. default: `1.2`.

문서의 폰트 이 클래스는 문서의 $\text{\X}\text{\LaTeX}$ 이나 $\text{\Lua}\text{\LaTeX}$ 으로 컴파일하면 라틴 문자는 TeX Gyre Pagella로, 한글은 Noto Serif CJK KR로 식자한다. `rmfamily`만 정의되어 있다. 따라서 소스 코드는 Latin Modern Typewriter와 UnDotum으로 식자된다. 문서 전체에 걸치는 폰트는 일반적인 `oblivoir` 방식으로 변경할 수 있다.

$\text{\pdf}\text{\LaTeX}$ 의 경우에는 표준 폰트만이 사용된다.

3.2 패키지 옵션

패키지만 별도로 사용할 때에는 다음과 같이 로드한다.

```
\usepackage[<options>]{srcandtext}
```

옵션은 다음과 같다. 이 옵션들은 클래스 옵션으로 지정해도 효과가 같다.

tcolorbox 이 값이 주어지면 verbatim 부분의 식자를 tcolorbox의 minted 옵션으로 식자한다. 이에 대하여는 아래 소절에서 조금 더 설명하겠다. default: false.

expl 2.2 소절에서 설명하는 바 모든 <sntsave>에 대하여 \autoexplon 상태로 설정하게 하는 옵션이다. default: false.

exampleenv \loadsnttext 명령을 사용했을 때 verbatim 부분을 디스플레이할 환경 이름을 적을 수 있다. default: exampleonly.

framecolor tcolorbox일 때 유효하다. 코드 리스팅하는 tcolorbox의 colframe을 설정한다. 기본값은 cyan. 클래스 옵션으로는 줄 수 없다.

backcolor tcolorbox일 때 유효하다. 코드 리스팅하는 tcolorbox의 colback을 설정한다. 기본값은 cyan! 10. 클래스 옵션으로는 줄 수 없다.

클래스 옵션인 bvlskip은 스타일에도 같은 방식으로 적용할 수 있다.

3.3 tcolorbox 옵션과 example 환경

tcolorbox 옵션이 주어지지 않으면 소스 코드를 verbatim으로 표시하는 방식이 <boxedverbatim>이다. 그러나 이 옵션이 주어지면 tcolorbox의 minted listing 방식으로 바뀐다.

<exampleside>과 <examplebelow>라는 환경이 정의되어 있다. 이것은 주어진 명령을 실행하여 그 결과를 소스 코드의 오른쪽이나 아래에 보이는 것이다. <exampleonly>는 단순히 소스 코드만을 보일 때에 쓰는 환경으로서 실제로 tcolorbox 옵션이 주어졌을 때 소스를 식자하는 환경인데, 필요에 따라 <verbatim> 대신 활용할 수 있다.

tcolorbox 옵션이 지정되지 않았을 때 이 환경들은 “listings”를 이용하여 리스팅하고 결과를 보인다. 그러나 tcolorbox 옵션이 주어지면 minted로 같은 결과를 만든다. 즉, tcolorbox 옵션이 있거나 없거나 tcolorbox에 의한 소스 코드 리스팅은 이루어진다. 이 옵션은 tcolorbox의 minted가 활성화되느냐 마느냐를 결정하는 것이다.

아무런 클래스 옵션도 지시하지 않으면 tcolorbox=false이다.

3.4 자잘한 기능들

편의 장치

이런 종류의 문서를 위하여 편의 장치를 마련하였다.

ShortVerb \MakeShortVerb를 적용하였다. | 부호만으로 verbatim을 식자할 수 있다. |verb|. verb.

\numpar 평문단의 왼쪽에 문단 번호를 붙이는 명령이다. 이 문단 번호는 \section에 의하여 reset된다. 리스트 문단(itemize, enumerate 따위)의 경우에는 잘 동작하지 않으므로 주의. \label을 붙여서 문단 번호를 참조할 수 있다. 예를 들면 이 문서의 뒷부분 예제에 붙인 \numpar \label{numpar:last}를 참조하면 다음과 같다: 8번 문단을 보시오. 이 참조 링크가 잘 동작하는지 확인하라.

\stm 인자로 주어지는 단어에 백슬래시를 붙여서 ttfamily로 식자한다. 그리고 이 문자열(매크로 이름)을 인덱스에 넣는다. 맨 처음 백슬래시를 붙이지 않아야 한다는 점에 주의. \stm{macroname}은 \macroname과 같이 식자된다.

`\ste` 주어지는 단어를 환경 이름으로 간주한다. 그리고 이를 인덱스에 넣는다. `\ste{envname}`은 `<envname>`과 같이 식자된다.

`\stp` 주어지는 단어를 패키지 이름이나 클래스 이름으로 간주한다. 인덱스에 넣는다. `\stp{pkgname}`은 `pkgname`과 같이 식자된다.

`\stopt` 주어지는 단어를 옵션 텍스트로 간주한다. 인덱스에 넣는다.

`\wi` 주어지는 단어를 평이하게 식자하고 인덱스에 넣는다. `\wi`의 인자는 주어진 대로 식자된다.

`\stm`, `\ste`, `\stp`에는 별표를 붙일 수 있다. 별표가 붙으면 명령의 끝에서 `\allowbreak`를 실행한다. 행 나눔의 문제가 있을 때 사용할 수 있다.

`tcolorbox`일 때 박스의 색상을 바꾸려면 `\sntcolorset` 명령을 쓸 수 있다. 두 개의 색상을 쉼표로 분리하여 지시한다.

```
\sntcolorset{black!80,yellow!20}
```

외부 파일

이 클래스는 `\jobname.code.snt`와 `\jobname.text.snt`라는 두 개의 부수 파일을 작업 폴더에 만든다. 전자에는 `<sntsave>`된 내용이 그대로 들어 있어서 “코드”로서 기능하고, 후자에는 주석을 제거하고 `verbatim` 텍스트로 코드를 배치하여 식자 가능한 “텍스트”로 기능하는 파일이다. 이 파일은 `<sntsave>` 환경이 실행될 때마다 갱신된다. 즉 이전 내용을 대체한다.

인덱스 만들기과 워크플로우 자동화

문서에 `\printindex`를 선언하면 찾아보기가 만들어진다.

이 안내문서에 사용한 인덱스 스타일 파일을 `srcandtext.ist`라는 이름으로 포함하였다. 이 파일은 다음 위치에 설치할 수 있다.

```
[$TEXMFHOME]/makeindex/srcandtext/srcandtext.ist
```

`ist` 파일은 `makeindex` 즉 `komkindex`와 함께 쓰기 위한 것이다.

```
$ komkindex -s srcandtext <filename>.idx
```

워크플로우 자동화 도구로 `arara`나 `spix`를 쓸 수 있다. `minted 3.0` 이전에는 `-shell-escape`가 필요했기 때문에 이러한 워크플로우 자동화 도구를 이용하는 것을 추천했다. 이 안내문서의 소스는 다음과 같이 시작한다.

```
%!TEX program = SpiX
%$ xelatex -synctex=1 $texname
%$ komkindex -k -s srcandtext.ist $basename.idx
%$ xelatex -synctex=1 $texname
```

`arara`라면 다음과 같이 할 수 있겠다.

```
%!TEX program = arara
% arara: xelatex: { synctex: yes }
% arara: komkindex: { koreanfirst: yes, style: srcandtext }
% arara: xelatex: { synctex: yes }
```


arara를 위한 komkindex rule 파일은 KTUG 사설저장소에서 arara-rules-ko라는 이름의 패키지로 내려 받을 수 있다.

4 예문

예제로서, 《예제로 배우는 Expl3》(공개판)의 p. 16의 3번 문제를 풀어보겠다.

문제

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 에러 처리 코드를 포함하라.

이어지는 문단부터 `<sntsave>`와 `\loadsnttext`를 이용한 결과이다.

Expl3 함수를 만드는 문제이다.

1. 필요한 변수를 선언한다. `\l_index_int`는 `tl`에 `map`을 걸 때 순서를 세기 위한 인덱스이다.

```
\int_new:N \l_index_int
```

2. 문제에 따라 명령을 정의한다.

```
\NewDocumentCommand \myitemswap { m m m m }
{
```

3. #2와 #3의 작은 값은 1 이상이어야 하고 큰 값은 #1 문자열의 길이 이하여야 한다. 일단 #2와 #3의 작은 쪽과 큰 쪽을 결정하여 작은 쪽을 `\l_tmpa_int`에 넣고, 큰 쪽을 `\l_tmpb_int`에 넣는다. 그런 다음 작은 쪽이 1 보다 작으면 무조건 1로 하고, 큰 쪽이 #1 문자열 길이보다 길다면 무조건 #1 문자열의 길이로 설정하자. 이렇게 하여 에러처리가 이루어졌다.

```
\int_set:Nn \l_tmpa_int { \int_min:nn { #2 } { #3 } }
\int_set:Nn \l_tmpb_int { \int_max:nn { #2 } { #3 } }
\int_compare:nT { \l_tmpa_int < 1 }
{
  \int_set:Nn \l_tmpa_int { 1 }
}
\int_compare:nT { \l_tmpb_int > \tl_count:n { #1 } }
{
  \int_set:Nn \l_tmpb_int { \tl_count:n { #1 } }
}
```

4. #2, #3 가운데서 숫자가 작은 쪽 아이템을 `\l_tmpa_tl`에 넣고, 큰 쪽 아이템을 `\l_tmpb_tl`에 넣을 것이다.

```

\tl_set:Nx \l_tmpa_tl { \tl_item:nn { #1 } { \l_tmpa_int } }
\tl_set:Nx \l_tmpb_tl { \tl_item:nn { #1 } { \l_tmpb_int } }

```

\tl_set:의 인자 형식이 :Nx로 된 이유는 \tl_item:nn을 확장하여 최종적인 ‘값’을 집어넣고 싶기 때문이다.

5. 교환은 다음과 같이 이루어진다. #1의 tl 아이템들을 하나씩 검토해나가다가, 인덱스 카운터가 “작은 쪽”과 같으면 \l_tmpb_tl을 ‘출력용 tl(#4)’에 넣고, “큰 쪽”과 같아지면 \l_tmpa_tl을 ‘출력용 tl’에 넣는 것이다.
6. \tl_map_inline:을 통해서 처리할 것이므로 아이템의 번호를 세기 위해서 인덱스 카운터를 초기화하고 출력용 tl을 비운다. 이 작업은 map에 들어가기 전에 한다.

```

\int_zero:N \l_index_int
\tl_clear:N #4

```

7. 이제 \tl_map을 돌리자. 만약 #1 인자를 재사용해야 한다면 이것을 \l_something_tl에 미리 받아두었다가 \tl_map_inline:Nn으로 map할 수 있다. 그러나 이 경우에 #1은 특별히 다른 목적에 쓰일 것 같지 않으므로 별도의 tl 변수에 할당하지 않고 진행하겠다. \tl_map_inline:nn.

```

\tl_map_inline:nn { #1 }
{

```

맨처음 하는 일은 인덱스 카운터를 1 증가시키는 것이다.

```

\int_incr:N \l_index_int

```

비교해야 할 숫자가 두 경우이므로 \int_case:nn이 적절한 해법이 된다.

```

\int_case:nnF { \l_index_int }
{
  { \l_tmpa_int } { \tl_put_right:NV #4 \l_tmpb_tl }
  { \l_tmpb_int } { \tl_put_right:NV #4 \l_tmpa_tl }
}

```

\tl_put_right:NV를 쓴 이유는 \l_tmpa_tl 등이 매크로 형식으로 들어오기 때문인데, v는 이 매크로를 그대로 넣지 말고 “그 값을 취해서” 넣으라는 의미이다.

\int_case:nnTF에서 T 부분에는 할 일이 없다. 아니, 할 일을 이미 적어주었다. 그래서 :nnTF가 아니라 :nnF로 T 부분을 생략하였다. F일 때, 즉 index = tmpa 또는 tmpb가 아닐 때 해야 할 일은 들어온 아이템을 그대로 출력 tl에 넣는 것이다. ##1은 무엇을 가리키는가? map에서 현재 들어오는 아이템이다.

```

{
  \tl_put_right:Nn #4 { ##1 }
}
}

```

위의 소스에서 마지막 종괄호는 \tl_map_inline:nn의 정의가 끝난 위치이다.

8. 여기까지 진행하면 #4 안에 원래의 #1 인자의 순서가 바뀐 아이템을 갖추게 된다. 명령의 정의를 종료한다.

```
}
```

이를 출력하려면 #4 매크로를 실행하면 되겠다.

이렇게 정의한 매크로가 잘 도는지를 검사하기 위하여 여기서 정의한 명령을 실행해보았다.

<pre>\myitemswap{catalina}{3}{6}{\myresult} \myresult</pre>	caialtna
---	----------

잘 되는 것을 알겠다. 코드가 필요하다면 \jobname.code.snt 파일의 내용을 본다. (다만 마지막 sntsave의 내용만이 들어 있음에 유의.) 이 파일의 내용을 <boxedverbatim>에 넣었다.

```
\ExplSyntaxOn %snt**
%% Expl3 함수를 만드는 문제이다.
%% \numpar 필요한 변수를 선언한다.
%% \stm{l_index_int}는 t1에 map을 걸 때 순서를 세기 위한
%% 인덱스이다.
%<<Code>
\int_new:N \l_index_int
%</Code>
%% \numpar 문제에 따라 명령을 정의한다.
%<Code>
\NewDocumentCommand \myitemswap { m m m m }
{
%</Code>
%% \numpar |#2|와 |#3|의 작은 값은 $1$ 이상이어야 하고
%% 큰 값은 |#1| 문자열의 길이 이하여야 한다.
%% 일단 |#2|와 |#3|의 작은 쪽과 큰 쪽을 결정하여 작은 쪽을
%% \stm{l_tmpa_int}에 넣고, 큰 쪽을 \stm{l_tmpb_int}에 넣는다.
%% 그런 다음 작은 쪽이 $1$보다 작으면 무조건 $1$로 하고,
%% 큰 쪽이 |#1| 문자열 길이보다 길다면 무조건 |#1| 문자열의 길이로
%% 설정하자. 이렇게 하여 예러처리가 이루어졌다.
%<Code>
\int_set:Nn \l_tmpa_int { \int_min:nn { #2 } { #3 } }
\int_set:Nn \l_tmpb_int { \int_max:nn { #2 } { #3 } }
\int_compare:nT { \l_tmpa_int < 1 }
{
\int_set:Nn \l_tmpa_int { 1 }
}
\int_compare:nT { \l_tmpb_int > \tl_count:n { #1 } }
{
\int_set:Nn \l_tmpb_int { \tl_count:n { #1 } }
}
%</Code>
%% \numpar |#2|, |#3| 가운데서 숫자가 작은 쪽 아이템을 \stm{l_tmpa_tl}에 넣고,
%% 큰 쪽 아이템을 \stm{l_tmpb_tl}에 넣을 것이다.
```

```

%<Code>
    \tl_set:Nx \l_tmpa_tl { \tl_item:nn { #1 } { \l_tmpa_int } }
    \tl_set:Nx \l_tmpb_tl { \tl_item:nn { #1 } { \l_tmpb_int } }
%</Code>
%% |\tl_set:|의 인자 형식이 |:Nx|로 된 이유는 \stm{tl_item:nn}을
%% 확장하여 최종적인 `값'을 집어넣고 싶기 때문이다.
%%
%% \numpar 교환은 다음과 같이 이루어진다. |#1|의 tl 아이템들을
%% 하나씩 검토해나가다가, 인덱스 카운터가 ``작은 쪽''과
%% 같으면 \stm{l_tmpb_tl}을 `출력용 tl(|#4|)'에 넣고, ``큰 쪽''과
%% 같아지면 \stm{l_tmpa_tl}을 `출력용 tl'에 넣는 것이다.
%%
%% \numpar \stm{tl_map_inline:}을 통해서 처리할 것이므로 아이템의
%% 번호를 세기 위해서 인덱스 카운터를 초기화하고 출력용
%% tl을 비운다. 이 작업은 map에 들어가기 전에 한다.
%<Code>
    \int_zero:N \l_index_int
    \tl_clear:N #4
%</Code>
%% \numpar 이제 |\tl_map|을 돌리자.
%% 만약 |#1| 인자를 재사용해야 한다면 이것을 |\l_something_tl|에
%% 미리 받아두었다가 \stm{tl_map_inline:Nn}으로 map할 수 있다.
%% 그러나 이 경우에 |#1|은 특별히 다른 목적에 쓰일 것 같지 않으므로
%% 별도의 tl 변수에 할당하지 않고 진행하겠다. \stm{tl_map_inline:nn}.
%<Code>
    \tl_map_inline:nn { #1 }
    {
%</Code>
%% 맨처음 하는 일은 인덱스 카운터를 $1$ 증가시키는 것이다.
%<Code>
        \int_incr:N \l_index_int
%</Code>
%% 비교해야 할 숫자가 두 경우이므로 \stm{int_case:nn}이 적절한 해법이
%% 된다.
%<Code>
        \int_case:nnF { \l_index_int }
        {
            { \l_tmpa_int } { \tl_put_right:NV #4 \l_tmpb_tl }
            { \l_tmpb_int } { \tl_put_right:NV #4 \l_tmpa_tl }
        }
%</Code>
%% \stm{tl_put_right:NV}를 쓴 이유는 \stm{l_tmpa_tl} 등이 매크로
%% 형식으로 들어오기 때문인데, |V|는 이 매크로를 그대로 넣지 말고
%% ``그 값을 취해서'' 넣으라는 의미이다.
%%
%% \stm{int_case:nnTF}에서 |T| 부분에는 할 일이 없다. 아니, 할 일을 이미 적어주었다.
%% 그래서 |:nnTF|가 아니라 |:nnF|로 |T| 부분을 생략하였다.
%% |F|일 때, 즉 index = tmpa 또는 tmpb가 아닐 때 해야 할 일은 들어온
%% 아이템을 그대로 출력 tl에 넣는 것이다.
%% |#1|은 무엇을 가리키는가? map에서 현재 들어오는 아이템이다.

```

```

%<Code>
    {
        \tl_put_right:Nn #4 { ##1 }
    }
%</Code>
%% 위의 소스에서 마지막 중괄호는 \stm{tl_map_inline:nm}의 정의가
%% 끝난 위치이다.
%%
%% \numpar \label{numpar:last} 여기까지 진행하면 |#4| 안에 원래의 |#1| 인자의
%% 순서가 바뀐 아이템을 갖추게 된다. 명령의 정의를 종료한다.
%<Code>
    }
%</Code>
%% 이를 출력하려면 |#4| 매크로를 실행하면 되겠다.
\ExplSyntaxOff %snt**

```

변경 이력

- version 0.9 (2025/02/13) minted 업데이트에 따른 수정.
- version 0.5 (2024/04/28) T_EXLive 2024의 xparse와 충돌하는 문제 해결. \sntcolorset 도입.
- version 0.4.3 (2020/11/06) \loadsnttext의 verbatim 환경의 이름을 지정할 수 있게 함.
- version 0.4.1 (2020/09/29) example 환경이 만드는 박스 내의 미묘한 간격 조절.
- version 0.4 (2020/09/25) 스타일이 표준 L^AT_EX 클래스 지원.
- version 0.3.3 (2020/09/24) math 옵션 동작을 수정.
- version 0.3.2 (2020/09/24) class와 style을 분리하였다.
- version 0.3 (2020/09/20) expl 옵션을 추가하였다.

찾아보기

【 □ 】

매크로

\allowbreak, 7
\autoexploff, 5
\autoexplon, 5, 6
\ExplSyntaxOff, 5
\ExplSyntaxOn, 4, 5
\input, 4, 5
\int_case:nn, 9
\int_case:nnTF, 10
\int_step..., 4
\int_step_inline:, 4
\l_index_int, 9
\loadsntcode, 4, 5
\loadsnttext, 4-6, 8
\loadsnttext*, 4, 5
\l_sum_int, 4
\l_tmpa_int, 9
\l_tmpa_tl, 9
\l_tmpb_int, 9
\l_tmpb_tl, 9
\MakeShortVerb, 7
\mysum, 4
\parindent, 6
\parskip, 6
\printindex, 8
\section, 7
\sntcolorset, 7
\ste, 7
\stm, 7
\stp, 7
\tl_item:nn, 9
\tl_map_inline:, 9
\tl_map_inline:Nn, 9
\tl_map_inline:nn, 9, 10
\tl_put_right:NV, 9
\wi, 7

문단 번호, 7

문학적 프로그래밍, 1, 2

【 ▢ 】

부수 파일, 8

【 ○ 】

여백 크기, 6

예제로 배우는 Expl3, 8

옵션

backcolor, 6

bvlskip, 6

colback, 6

colframe, 6

exampleenv, 6

expl, 5, 6, 12

extraopt, 6

framecolor, 6

margin, 6

math, 6, 12

paper, 6

parindent, 6

parskip, 6

tcolorbox, 6, 7

옵션 텍스트, 7

워크플로우 자동화, 8

인덱스, 7

인덱스 스타일 파일, 8

【 × 】

주석문, 3

【 ㄹ 】

찾아보기, 8

【 ㄴ 】

클래스 이름, 7

【 ㅍ 】

패키지

amsmath, 6

arara, 8

fapapersize, 6

listings, 7

minted, 6-8, 12

ob-mathleading, 6

oblivoir, 6

spix, 8

srcandtext, 1

srcandtext.cls, 5

stcandtext.sty, 5

tcolorbox, 1, 2, 5, 7

패키지 이름, 7

편의 장치, 7

폰트, 6

【 ㅎ 】

환경

boxedverbatim, 6, 7, 10
examplebelow, 7
exampleonly, 7
exampleside, 7
sntsave, 2, 4–6, 8
*sntsave**, 5
verbatim, 1, 7
 환경 이름, 7

【A】

\allowbreak, 7
amsmath, 6
arara, 8
\autoexploff, 5
\autoexplon, 5, 6

【B】

backcolor, 6
boxedverbatim, 6, 7, 10
bvlskip, 6

【C】

colback, 6
colframe, 6

【E】

examplebelow, 7
exampleenv, 6
exampleonly, 7
exampleside, 7
expl, 5, 6, 12
expl3 프로그래밍, 1
expl3 함수, 1
*expl3*의 for loop, 4
\ExplSyntaxOff, 5
\ExplSyntaxOn, 4, 5
extraopt, 6

【F】

fapapersize, 6
framecolor, 6

【I】

\input, 4, 5
\int_case:nn, 9
\int_case:nnTF, 10
\int_step_..., 4
\int_step_inline:, 4

【L】

\l_index_int, 9
listings, 7
\loadsntcode, 4, 5

\loadsnttext, 4–6, 8
*\loadsnttext**, 4, 5
\l_sum_int, 4
\l_tmpa_int, 9
\l_tmpa_tl, 9
\l_tmpb_int, 9
\l_tmpb_tl, 9

【M】

\MakeShortVerb, 7
margin, 6
math, 6, 12
minted, 6–8, 12
\mysum, 4

【O】

ob-mathleading, 6
oblivoir, 6

【P】

paper, 6
papersize, 6
\parindent, 6
parindent, 6
\parskip, 6
parskip, 6
preamble, 5
\printindex, 8

【S】

\section, 7
\sntcolorset, 7
sntsave, 2, 4–6, 8
*sntsave**, 5
spix, 8
srcandtext, 1
srcandtext.cls, 5
stcandtext.sty, 5
\ste, 7
\stm, 7
\stp, 7

【T】

tcolorbox, 1, 2, 5, 7
tcolorbox, 6, 7
\tl_item:nn, 9
\tl_map_inline:, 9
\tl_map_inline:Nn, 9
\tl_map_inline:nn, 9, 10
\tl_put_right:NV, 9

【V】

verbatim, 1, 7

【W】
\wi,7