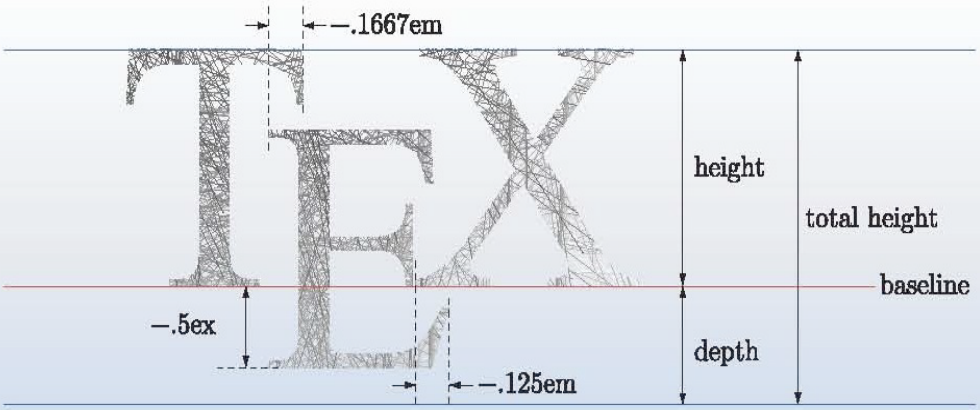
April, 2006

KTUG Faq에 올라온 읽을 만한 글을
모아서 엮은 책. 2006년 4월.

KTUGFaq



*The TEXbook*에 정의된 TEX 로고 명령.

```
\def\TeX{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX}
```

이 로고의 구현은 TEX이 단순한 문서작성기를 뛰어넘은 조판 프로그램임을 보여준다.

차 례

Editorial: TEX에 새롭게 관심이 집중되고 있습니다. 이번 달에 올라온 글 중에서 작은나무 님이 연재하는 '*TEXbook*을 읽는다'[1]의 중간 결산인 "매크로 작성 기법"과 김도현 교수님의 "초미니 한글 plainTEX"을 골라서 싣습니다.

1 매크로 작성 기법

작은 나무[2]

그동안 plainTEX을 공부하면서 익힌 매크로 작성 기법을 주제로 문서를 만들어 보았습니다. 계속 수정해나갈 것이며, 새로운 것을 익히면 추가해 나갈 계획입니다.

plainTEX 연습 겸 LATEX이 아닌 plainTEX으로 작성하였습니다. :)

1) http://faq.ktug.or.kr/faq/LittleTree/ReadingTeXbook

2) http://faq.ktug.or.kr/faq/LittleTree/ReadingTeXbook/2006-04#23

TEX 매크로 작성 기법

작은나무

TEX이 워드프로세서와 뚜렷이 구별되는 점은 프로그래밍이 가능하다는 점이다. 그리고 그 프로그래밍의 핵심에는 바로 "*매크로*(*Macro*)"가 있다. 워드프로세서에도 매크로 기능이 있지만, TEX의 매크로는 워드프로세서의 그것과는 차원이 다르다. 간단해 보이는 매크로 하나로 환상적인 여러 가지 기능이 이루어지는 것을 보고 있노라면 매크로를 익히고자 하는 유혹에 빠지지 않을 수 없다. 그런데 매크로를 작성하는 기법을 익히는 것이 그리 쉬운 것은 아니다. 혹시 여유가 있거나 호기심이 있다면 plain TEX의 포멧 파일인 `plain.tex`을 열어보라. 마치 암호문 같을 것이다. 하지만 보기에만 그렇다. 그 매크로에 쓰이는 primitive 명령어들을 하나씩 익혀가다 보면, 아무리 어렵고 복잡한 매크로라 할지라도 그 정체를 곧 드러내기 마련이다. 하지만 애석하게도 primitive 명령어를 모두 안다고 해서 매크로를 곧바로 작성할 수 있는 것은 아니다. 어린 아이가 한글을 모두 안다고 해서 모든 책을 읽고 이해 할 수는 없는 것과 마찬가지이다. 매크로를 작성하기 위해서는 TEX의 기본적인 primitive 명령어들을 익히고, 매크로 작성 기법을 익혀야 한다. 작은나무가 살펴 본 바로는 매크로 작성 기법에는 일종의 패턴이 있다. 그 패턴을 익히고, 외우면 비교적 손쉽게 TEX의 매크로를 작성 할 수 있다. 그리고 TEX의 매크로를 알아가면 갈수록 그만큼 더 TEX의 매력에 빠지게 된다.

이 글은 plain TEX에서 간단한 매크로를 만들어 본 경험이 있고, TEX에 대한 약간의 지식이 있으나, 매크로의 위력에 반해 매크로를 작성하고자 하는 사람들을 위한 글이다. 따라서 이 글을 읽는 이는 적어도 매크로 작성에 자주 사용되는 `\global`, `\outer`, `\long`, `\def`, `\edef`, `\let`, `\futurelet`, `\if`, `\expandafter`, ... 등을 한 번쯤 관심있게 본 적이 있다고 가정한다. 또한 이 글은 작은나무가 TEX 공부를 하다가 느낀 어려운 점들을 작은나무와 같은 초보자의 눈높이에서 쓴 글이기도 하다.

이 글은 Victor Eijkhout의 *TEX by Topic*과 Amy Hendrickson의 *Getting TEXnical: Insights into TEX Macro Writing Techniques*, TUGboat, Volume 11 (1990), No. 3—1990 Conference Proceedings 을 많이 참고하였고, 특히 일부는 이 두 문서를 번역하였다.

꼬리 재귀 기법 (Tail recursion)

꼬리 재귀 기법은 주로 매크로의 인자를 구성하는 토큰들을 하나씩 처리 하는 데에 주로 사용된다. 예를들어, `\foo`라는 매크로가 꼬리 재귀 기법으로 작성된 매크로라고 할 때, '`\foo{AbcDEFghIj}`'라고 하면, 인자로 주어진 '`AbcDEFghIj`'가 '`A`', '`b`', '`c`', '`D`', '`E`', ... 와 같이 토큰들로 분해되어, 하나의 문자 토큰 각각에 매크로 `\foo`가 적용되는 경우이다. 이처럼 각각의 토큰에 적용되는 꼬리 재귀 기법은 다른 많은 기법의 기본이 되기도 하고, 이를 응용한 매크로 역시 매우 많다. 즉, 각각의 문자 토큰에 적용된다는 개념을 이용하여, 각각의 토큰들이 아니라 각각의 단어에 적용하기도 한다. 이 기법은 주어진 매크로가 어떤 조건 하에서 인자의 토큰에 하나씩 적용되므로 "조건, 반복"에 대한 명령어가 주로 사용된다. 따라서 `\let`, `\if`, `\expandafter` 등의 명령어에 대한 지식이 요구된다. 이 기법은 대개 아래와 같은 패턴을 가지고 있다.

```
\def\foo#1{\bar#1\stop}
\def\bar#1{\ifx#1\stop \let\next=\relax
  \else Do Something \let\next=\bar\fi \next}
```

이 패턴은 너무도 많이 쓰이고 유용하기 때문에 익히는 정도가 아니라 반드시 외어 두어야 한다. 하나씩 자세히 살펴보자. 먼저 `\foo`가 다음과 같이 정의되었기 때문에,

```
\def\foo#1{\bar#1\stop}
```

`\bar`의 인자는 원래 주어진 인자 `#1`에 `\stop`를 더한 합이 된다. 여기서 `\stop`은 매크로 인자의 구분자(delimiter)로 사용되었기 때문에 정의 되어있지 않아도 상관없다. 여기서 `\stop`은 주어진 원래의 인자 바로 다음에 붙어서 인자의 마지막을 가리키는 토큰이 된다. 따라서 `\bar`가 토큰들을 하나씩 처리하다가 `\stop`을 만나면, 그 실행을 멈추게 된다. 위의 매크로 정의에서 실제로 일을 하는 매크로는 `\bar`이고, `\foo`는 `\bar`가 처리하는

과정에서 어디까지 실행하라는 그 끝을 알려주는 역할을 한다. \bar의 정의를 보면 알 수 있듯이, \bar가 하는 일은 인자의 토큰을 하나씩 처리하면서 다음에 처리 할 토큰이 \stop이면, 더 이상 처리할 토큰이 없으므로, 다음에 실행할 명령어 \next가 아무 일도 하지 말라는 뜻의 primitive, \relax가 되어 그곳에서 실행을 멈추고, 처리 할 토큰이 \stop이 아니면 \next가 \bar가 되어 그 다음 토큰을 처리하게 된다. 즉, \bar가 자기 자신을 호출하는 재귀적 방법으로 반복을 구현하고 있다는 것을 확인 할 수 있다.

여기서 눈여겨 보아야 할 것은 \let의 사용법 이다. 만약 \bar가 다음과 같이 정의 되었다면, 무엇이 잘못된 것일까?

```
\def\bar#1{\ifx#1\stop \relax \else Do Something \bar\fi}
```

매크로 \bar는 하나의 인자를 갖는데, 위와 같은 정의의 치환문(replacement text)에서는 \fi가 바로 \bar의 인자가 되어버려서 전개가 엉망이 된다. 주의가 필요한 대목이다. "나는 \let을 죽어도 사용하기 싫다." 하는 이는 \expandafter를 사용하여 다음과 같이 하면 된다.

```
\def\bar#1{\ifx#1\stop \expandafter\relax
  \else Do Something \expandafter\bar\fi}
```

위와 같은 정의가 가능한 이유는, \expandafter의 성질 상 \bar를 곧 바로 전개하는 것이 아니라 그 다음에 나오는 \fi를 처리한 다음에 \bar가 실행 되어 \fi가 \bar의 인자가 되는 경우가 없기 때문이다. 위의 매크로 정의에서 \expandafter는 \fi를 제거하는 역할을 한다. 이 방법 또한 대단히 유용하고 익혀 둘만한 기법이기는 하지만 \let을 사용하는 경우가 대부분이고 그것이 기본이다.

이제 패턴을 알았으니, 몇 가지 예를 통해서 이 패턴의 사용법을 익혀보자. 먼저 TEXbook, 219쪽에 나와있는 예로, 주어진 인자에서 공백이 아닌 토큰의 갯수를 세는 \length 매크로를 만들어 보자. '\length{abcd efg}'는 공백이 아닌 토큰의 갯수가 7이므로 '7'을 출력하고, \length{argument}는 '8'을 출력한다. 보다 이해를 쉽게하기 위해서, 이 패턴을 하나도 바꾸지 말고 매크로 이름만 바꿔서 그대로 적용해 보자. 위의 패턴에서 \foo에 해당하는 매크로를 \length로 하고, \bar에 해당하는 매크로를 \getlength로 바꾸고 나머지는 그대로 두면 \length 매크로는 다음과 같은 모양을 하게 된다.

```
\def\length#1{\getlength#1\stop}
\def\getlength#1{\ifx#1\stop \let\next=\relax
  \else Do Something \let\next=\getlength\fi \next}
```

위에 정의한 그대로 '\length{argument}'를 실행하면 결과는 다음과 같다.

Do Something Do Something Do Something Do Something Do Something Do Something Do Something Do Something

위 결과에서 알 수 있듯이 'Do Something'이 8번 출력되었다. 즉, \getlength가 여덟 번 실행 되었다는 것이다. 이는 인자 중에서 공백이 아닌 토큰의 갯수가 여덟 개이기 때문에 그렇다. 그렇다면 해야 할 일이 분명해졌다. 'Do Something' 부분을 '8' 이라는 숫자를 출력 할 수 있게끔 해주는 무언가로 바꾸기만 하면 된다. 이 작업은 매우 간단하여 'Do Something'을 '\advance\count0 by1'로 바꾸기만 하면 됩니다. \advance\count0 by1이 하는 일은 \count0의 값을 1씩 증가시키는 일을 한다. 따라서 \count0의 값을 0으로 초기화 한 후에 1씩 증가시키는 작업을 여덟 번 수행하면 \count0의 값은 8이 된다. 여기서 \length 매크로가 \count0의 값을 0으로 초기화하고, 그 결과 8을 출력하는 역할을 한다.

```
\def\length#1{{\count0=0 \getlength#1\stop \number\count0}}
\def\getlength#1{\ifx#1\stop \let\next=\relax
  \else\advance\count0 by1 \let\next=\getlength\fi \next}
```

실행해 보면, '\length{argument}'의 결과는 '8'이 나온다.

다른 예를 들어보자. 이 번에는 주어진 인자의 문자 토큰들 중에서 대문자의 갯수를 세는 매크로를 한 번 만들어 보겠다. 패턴은 위와 동일하다. 따라서 해결 방법도 위와 매우 유사하고 쉽다. 단지 'Do Something' 부분에 대문자인지 확인을 하고, 대문자이면, \length 매크로에서 처럼 \count0의 값을 하나 증가 시키면 된다. 즉 \length 매크로와 똑같다. 다만 차이점은, 대소문자 구분하는 루틴이 더 들어간다는 점이다.

```
\def\countucletter#1{{\count0=0 \ucact#1\stop \number\count0}}
\def\ucact#1{\ifx#1\stop \let\next=\relax
  \else\uppercase{\if#1}#1\advance\count0 by1 \fi\let\next=\ucact\fi \next}
```

매크로 \getlength와 유일한 차이점 이면서, 눈여겨 봐야 할 것이 대소문자 판별하는 부분이다.

```
\uppercase{\if#1}#1
```

이 방법 역시 기억해 둘 만한 기법이다. \uppercase는 주어진 인자중 소문자를 대문자로 바꾸는데, 명령어(control sequence)는 변경하지 않는다. 즉 \if는 명령어 이므로 \uppercase의 영향을 받지 않아서, 위의 문장은 결국 다음과 같다.

```
\if#1#1
```

첫번째 #1은 대문자로 바뀐 것이 두번째의 #1는 원래 그대로 이므로, 원래의 #1이 대문자이면 \if 문장이 참이 될 것이고, 소문자 였다면, \if의 값은 거짓이 되는 원리이다.

이 번에는 좀 더 응용하여 인자 각각의 문자 토큰에 적용되는 것이 아니라, 공백으로 구분되는 단어에 적용되는 경우를 살펴보자. \underlinewords'라는 매크로를 정의 하는데, 이 매크로가 하는 일은 주어진 단어에 모두 밑줄을 긋는 것이다.

```
\underlinewords
This is a handbook about \TeX, a new typesetting system intended for ...*
```

이 결과는 다음과 같다.

This is a handbook about TEX, a new typesetting system intended for ...

위에서 보면 '*'가 꼬리 재귀 기법의 설명에서 \stop이 된다는 것을 알 수 있을 것이다. 구현을 살펴보자. 아마 꼬리 재귀 패턴과 너무도 유사하다는 것을 금방 눈치 챌 수 있을 것이다.

```
\def\aster{*}
\long\def\underlinewords#1*{%
  \def\wstuff{#1 }\leavevmode\expandafter\ulword\wstuff * }
\long\def\ulword\#1 {\def\one{#1}%
  \ifx\one\aster \let\next\relax
  \else\vtop{\hbox{\strut#1}\hrule \relax} \let\next\ulword\fi \next}
```

이 매크로에서 꼬리 재귀 기법에서의 입력 토큰의 마지막을 나타내던 \stop의 역할을 하는 토큰은 \aster 이고, 'Do Something'은 '\vtop{\hbox{\strut#1}\hrule \relax}'에 해당한다. 그렇다면 도대체 위의 어떤 부분이 인자를 토큰 단위가 아닌 단어 단위로 처리하게 한다는 말인가? 그 해답은 바로 '␣'에 있다. 눈 여겨 보아야 할 부분이 바로 '\long\def\ulword#1 {\def\one{#1}%'이다. 바로 이 부분이 단어 단위로 처리하도록 해준다. '\ulword#1' 다음의 공백 '␣'은 매우 중요해서, 공백을 거의 무시하는 TEX일지라도 이 경우만은 그렇지 않다. 매개변수 뒤에 나오는 공백은 구분자로 사용되므로 무시하면 안된다. 이 공백이 하는 역할이 결정적이어서 \ulword 매크로는 토큰 단위가 아닌 공백을 구분자로 하는 단어 단위, 즉 'This␣', is␣', 'a␣', 'handbook␣', 'about␣', ... 등을 그 단위로 취급한. 상황이 이러하다보니, \stop로 사용되는 * 뒤에도 공백이 필요하여,'{... \wstuff␣*␣}'와 같은 모양이 나온 것이다. 반드시 기억 해 둘만한 테크닉이다.

좀 더 응용해 보자. 이 꼬리 재귀 기법을 이용하면 인자의 갯수가 변하는 매크로도 간단히 작성할 수 있다. 매크로를 작성하다 보면, 종종 그 인자의 갯수를 미리 알 수 없는 경우가 있다. 인자의 갯수를 알 수 없다는 말은 인자의 갯수가 변한다는 말과 같은 뜻 일것이다. 체스에서 말들을 움직이는 문제를 예를 들어서 생각해 보자. 체스 말의 위치를 나타내는 문자열들을 인자로 받는 다음과 같은 매크로는,

```
\White(Ke1,Qd1,Na1,e2,f4)
```

아래와 같은 순서의 명령들을 나열해 놓은 것과 같다.

```
\WhitePiece{K}{e1} \WhitePiece{Q}{d1} \WhitePiece{N}{a1}
\WhitePiece{P}{e2} \WhitePiece{P}{f4}
```

여기서 체스에서 장기의 졸(卒)을 나타내는 'P'는 \White에서 생략된 것을 주목하자.

매크로 \White를 직접 정의해 보자. 먼저 첫번째로 해결해야 할 문제는 \White 매크로의 인자의 갯수가 변할 수 있다는 점이다. 이 것을 해결하기 위해서 꼬리 재귀 기법에서 사용했던 인자의 끝을 나타내는 구분자를 사용하는 테크닉을 이용하자.

```
\def\White(#1){\xWhite#1,xxx,}
\def\endpiece{xxx}
```

꼬리 재귀 기법에서 익힌 그대로 이다. 이상할 것 하나도 없다. \xWhite 매크로는 다음과 같이 정의 된다.

```
\def\xWhite#1,{\def\temp{#1}%
  \ifx\temp\endpiece
  \else \WhitePieceOrPawn#1XY%
        \expandafter\xWhite
  \fi}
```

이것 역시 이상할 것 하나 없다. 다만 \let을 이용하는 대신 \expandafter가 사용됐다는 것이 다른 점인데, 이 역시 꼬리 재귀 기법에서 익힌 그대로 이다.

위에서 예를 든 매크로의 인자에서 보면 알 수 있듯이, 말의 위치를 나타내는 문자열의 길이는 둘 또는 셋이다.('Ke1' 혹은 'e2') 이 둘을 구분하는 방법은 \WhitePieceOrPawn의 인자를 네개로 만드는 것이다 즉 다음과 같은데,

```
\def\WhitePieceOrPawn#1#2#3#4Y{
    \if#3X \WhitePiece{P}{#1#2}%
    \else \WhitePiece{#1}{#2#3}\fi}
```

문자열의 길이가 3일때, 즉 예를 들면 Ke1일때, \xWhite 정의에서 \WhitePieceOrPawn#1XY라고 했으므로 결국은 \WhitePieceOrPawn Ke1XY가 되고, 문자열이 2일때, 예를 들어 e2일때는 \WhitePieceOrPawn e2XY가 된다. 따라서 문자열의 길이가 2이면 #3에 해당하는 토큰은 언제나 X가 되므로 #3의 값이 X인지 아닌지를 알면 문자열의 길이가 2인지 3인지 알 수 있다. 그것이 바로 매크로 \WhitePieceOrPawn의 정의이다.

인자 검사하기 (Examining the argument)

매크로를 작성하다보면, 주어진 매크로의 인자가 어떤 특정한 요소를 가지고 있는지 검사해야할 경우가 있다. 실제 예로 다음을 한번 생각해보자. 어떤 글의 제목과 저자가 다음과 같이 주어지는 매크로를 가정하자.

```
\title{An angle trisector}
\author{A.B. Cee\footnote*{Research supported by the
Very Big Company of America}}
```

저자가 두 명 이상일때는 다음과 같다.

```
\author{A.B. Cee\footnote*{Supported by NSF grant 1}
        \and
        X.Y. Zee\footnote{**}{Supported by NATO grant 2}}
```

매크로 \title과 \author는 다음과 같이 정의되어 있다고하자.

```
\def\title#1{\def\TheTitle{#1}}  \def\author#1{\def\TheAuthor{#1}}
```

그리고, 위 매크로는 다음과 같은 식으로 사용된다고 하자.

```
\def\ArticleHeading{ ... \TheTitle ... \TheAuthor ... }
```

어떤 저널은 기사의 저자와 제목을 모두 대문자로 표기하기도 한다. 그러한 경우의 구현은 다음과 같다.

```
\def\ArticleCapitalHeading
  { ...
   \uppercase\expandafter{\TheTitle}
    ...
   \uppercase\expandafter{\TheAuthor}
    ...
  }
```

위에서 \expandafter는 그 성질상 두번째 인자를 먼저 전개시키므로 \TheTitle과 \TheAuthor가 전개된다. (\expandafter의 첫번째 인자는 '{'이다.) 따라서 \expandafter에 의해서 전개되고난 결과를 \uppercase에 적용하므로 \TheTitle과 \TheAuthor 모두 다 대문자로 바뀐다. 그러나 \TheAuthor의 경우에는 한가지 문제점이 있다. footnote까지 모두 대문자로 바뀐다는 점이다. 이것을 해결해 보자.

먼저 저자가 한 명인 경우를 다루어보자. 그러면 다음과 같은 방법으로 해결 할 수 있다.

```
\expandafter\UCnoFootnote\TheAuthor
```

위의 것은 다음과 같이 전개된다.

```
\UCnoFootnote A.B. Cee\footnote*{Supported ... }
```

그리고 \UCnoFootnote를 다음과 같이 정의하면,

```
\def\UCnoFootnote#1\footnote#2#3{\uppercase{#1}\footnote{#2}{#3}}
```

위의 매크로는 다음과 같이 정확하게 인자가 매치 된다.

```
#1<-A.B. Cee
#2<-*
#3<-Supported ...
```

하지만, 이 매크로의 경우에는 footnote가 없다면, 완전히 잘못 동작할 것이다. 다음과 같이 수정해보자.

```
\expandafter\UCnoFootnote\TheAuthor\footnote 00
\def\stopper{0}
\def\UCnoFootnote#1\footnote#2#3{\uppercase{#1}\def\tester{#2}%
  \ifx\stopper\tester
  \else\footnote{#2}{#3}\fi}
```

위 매크로는 footnote가 없으면 정상 동작하나 하나라도 있다면, 오동작한다. 뭐가 문제인가? 위 매크로를 보면 꼬리 재귀 기법과 유사해 보인다. 그렇다. 꼬리 재귀 기법을 적용하면 footnote가 있건 없건 하나 있건 여러 개 있건 상관 없이 동작한다.

```
\def\stopper{0}
\def\UCnoFootnote#1\footnote#2#3{\uppercase{#1}\def\tester{#2}%
  \ifx\stopper\tester
  \else\footnote{#2}{#3}\expandafter\UCnoFootnote
  \fi}
```

위 꼬리 재귀의 경우는 \let을 사용하지 않고, \expandafter를 이용해서 해결하였다.

\futurelet을 이용한 옵셔널 인자

먼저 \let과 \futurelet의 의미부터 확실히 하자. 다음의 \let을 이용한 문장에서는

```
\let<control sequence><token1><token2><token3><token...>
```

<control sequence>가 <token1>의 의미를 갖게 되고, 남은 것을 다시 써보면, 다음과 같이 된다.

```
<token2><token3><token...>
```

즉, 위에서 <token1>은 <control sequence>가 되어 사라진다. 하지만 \futurelet은 좀 다르다. 위에서 \let을 \futurelet으로 바꾸면, 다음과 같이 되는데,

```
\futurelet<control sequence><token1><token2><token3><token...>
```

이는 <control sequence>에 <token1>이 아닌 <token2>의 의미를 부여한다. 그리고나서, \let의 경우는 그 의미상 <token1>이 사라진 반면, \futurelet에서는 <token2>가 사라지지 않는다.

```
<token1><token2><token3><token...>
```

위 상태가 되고나면, <token1>은 <token2>가 어떤 녀석인지 알게된다. 그래서 <token1>은 <token2>에 따라서 행동을 취하면 된다. 이제 \futurelet의 의미가 확실히 이해가 된다. 이놈을 어디다 쓰느냐가 문제인데... \futurelet이 쓰이는 대표적인 예가 바로 매크로의 파라미터가 옵셔널한 경우라고 한다. 다음과 같은 매크로 \Com이 있다고 하자.

```
\Com{argument}
\Com[optional]{argument}
```

위에서 처럼 \Com 이라는 명령어에 optional 이라는 파라미터가 있을 수도 있고, 없을 수도 있다. 즉 말 그대로 파라미터가 옵션이라는 뜻이다. 위와 같은 \Com 매크로를 해결하기 위해서 \futurelet이 주로 쓰인다고 한다. 그럼 \Com은 실제로 어떻게 정의 될까?

```
\def\Com{\futurelet\testchar\MaybeOptArgCom}
\def\MaybeOptArgCom{\ifx[\testchar \let\next\OptArgCom
  \else \let\next\NoOptArgCom \fi \next}
\def\OptArgCom[#1]#2{ ... }
\def\NoOptArgCom#1{ ... }
```

모든 것이 분명해지는 순간이다. 결국 아래의 문장에서

```
\Com[optional]{argument}
```

위의 \Com의 정의대로라면,

```
\futurelet\testchar\MaybeOptArgCom[optional]{argument}
```

이 되는데, \futurelet의 정의에 따라 \testchar는 '['를 갖게 된다. 그래서 \MaybeOptArgCom은 이제 \testchar이 어떤 놈인지 알게되어 \testchar가 '['라면 \OptArgCom를 실행하면되고, 아니면 \NoOptArgCom를 전개하면 그만이다.

재미있는 \csname

\csname의 유용한 기능 중의 하나는 \csname...\endcsname 내에서 control sequece들이 전개될 수 있다는 것이다. 이 말은 매우 중요한 의미를 내포하고 있다. 즉 control sequence의 이름이 주어진 조건에 따라서 동적으로 변할 수 있다는 뜻이다:

```
\expandafter\def\csname\testmacro\endcsname{<definition>...}
```

카운터 레지스터도 사용될 수 있다:

```
\expandafter\def\csname\testcounter\endcsname{<definition>...}
```

로마 숫자를 가진 카운터 레지스터도 사용될 수 있다:

```
\expandafter\def\csname\romannumeral\testcounter\endcsname{<definition>...}
```

심지어는 control sequence의 이름으로 사용될 수 없는, 문자 이외의 기호나 숫자들도 control sequence의 이름이 될 수 있다.

```
\expandafter\def\csname 123&#\endcsname{<definition>...}
```

여기서 한가지 기억해야 할 것은, 이처럼 문자가 아닌 숫자나 다른 기호들로 이루어진 control sequence들은 호출될 때도 정의 할때와 마찬가지로, 반드시 \csname 123&#\endcsname와 같은 형태로 호출 되어야 한다는 것이다. \expandafter와 함께 \csname를 사용하면, 다른 방법으로는 구현할 수 없는 재미있는 일들을 할 수 있다.

예를 들어보자. 어떤 한 매크로가 있고, 이 매크로는 또다른 한 매크로를 정의하고, 이 두번째 매크로의 이름은 주어진 상황에 따라서 변할 수 있다고 하자. 다음의 예에서 매크로 \usearg는 주어진 인자들 중에서 처음 두개의 단어를 각각 인자 #1와 #2로 받아서, 그 둘의 순서를 바꿔서 또다른 하나의 control sequence의 이름으로 하는 매크로를 정의한다. 이렇게 정의된 매크로는 이 매크로의 이름으로 정의된 사람의 이름과 주소를 나타내는데 사용한다고 하자. 여기서 이름의 순서를 바꾸는 이유는 그 이름들은 보조 파일로 들어가서 나중에 이름 순으로 정렬을 하기 위함이다. (Donald Knuth가 인덱스에서는 Knuth, Donald로 나오는 것과 같다.) 그 이름 순으로 정렬된 보조 파일을 잘 활용하면, 메일링 리스트를 만들 수도 있다. 매크로 \usearg를 자세히 살펴보자. 다음에서 \obeylines이 하는 역할은 라인의 끝을 나타내는 문자 ^^M의 카테고리 코드를 13으로 만들어서 매크로 인자의 구분자로 사용할 수 있도록 하고, ^^M을 \par로 정의한다.

```
{\obeylines
\def\usearg#1 #2^^M#3^^M^^M{%
  \expandafter\gdef\csname #2#1\endcsname
  {#1 #2\par #3}}
\usearg George Smith
21 Maple Street
Ogden, Utah 68709
\SmithGeorge
}
```

결과는 다음과 같다.

George Smith
21 Maple Street
Ogden, Utah 68709

이 기법에 대한 자세한 예제는 참고문헌 [3]과 그의 부록에 완벽한 예와 함께 자세히 설명되어 있다. 반드시 참고하기 바란다.

투스텝 매크로: 알고보면 두 단계로 이루어진 매크로

어떤 매크로는 사실 알고보면 두 개의 매크로로 이루어진 경우가 있다. 이러한 경우 한 매크로가 대부분의 일을 하고 다른 한 매크로가 그 매크로가 실행하는데 필요한 여러가지 환경을 초기화 한다. 이러한 매크로를 투스텝 (Two-step) 매크로 라고 한다.

예를 들어보자. 라인의 끝을 구분자(delimeter)로 갖는 인자를 취하는 '`\PickToEol`' 이라는 매크로가 있다고하자. 투 스텝의 첫번째 단계로 인자가 없는 매크로로 라인의 끝에 해당하는 문자의 카테고리 코드를 변경하고나서, 두번째 단계의 매크로를 호출하는 매크로는 다음과 같다.

```
\def\PickToEol{\begingroup\catcode`\^^M=12 \xPickToEol}
```

두번째 단계의 매크로는 라인의 끝까지 모든 토큰들을 하나의 인자로 받을 수 있다.

```
\def\xPickToEol#1^^M{ ... #1 ... \endgroup}
```

이 매크로 정의에는 한 가지 문제점이 있다: `^^M` 문자의 카테고리는 12 이어야 한다는 것입니다. 따라서 이를 바로잡아 매크로를 다시 정의하면 다음과 같다.

```
\def\PickToEol{\begingroup\catcode`\^^M=12 \xPickToEol}
{\catcode`\^^M=12 %
 \gdef\xPickToEol#1^^M{ ... #1 ... \endgroup}%
}
```

여기서 `^^M`의 카테고리 코드의 값은 `\xPickToEol`을 위해서 변경되었다. 주목해야 할 것은 `\PickToEol`에 사용된 `^^M`은 control symbol로 쓰였기 때문에 카테고리 코드와는 상관이 없다는 것이다.

주석 처리 환경

첫번째 기법인 꼬리 재귀 기법과 방금 전에 익힌 투스텝 매크로 기법을 응용한 예로써, 주석 처리 환경을 이루는 매크로를 만들어 보자.

TEX을 이용해서 문서를 만들다 보면, 가끔 디버깅 혹은 테스트 목적으로 텍스트의 일정 부분을 주석처리 해야할 경우가 있다. 그 주석처리 해야할 곳이 작다면 매 줄마다 `%`로 시작해서 해결되겠지만, 그 양이 무척 많다면 '`%`'도 무용지물이 될 것이다. 그래서 다음과 같은 명령어가 절실히 필요하다.

```
\comment
...
\endcomment
```

이것은 어떻게 구현하면 될까? 가장 간단한 방법은 아마 다음과 같을 것이다.

```
\def\comment#1\endcomment{}
```

위 구현은 간단하면서도 매우 참신한 아이디어 이지만, 몇 가지 결점을 가지고 있다. 예를들면, 주석 처리할 부분에 outer 매크로가 들어있거나 짝이 맞지 않는 괄호들이 있을 때는 위 방법으로 안된다는 것이다. 하지만,

무엇보다도 가장 큰 단점은 TEX이 위의 \comment를 처리할 때 #1를 모두 인자로 받아들인다는 점이다. TEX은 인자들을 처리하기 위해서 내부적으로 버퍼를 가지고 있을 텐데, #1이 무지하게 크고 많다면, 그 버퍼가 다 차버려서 TEX이 작업할 공간이 없어져서 에러 메시지를 뿌려댈 것이다.

가장 큰 단점, 버퍼가 부족할 수 있다는 단점을 해결하는 방법은 #1을 한번에 처리하지 말고, 라인 단위로 처리하면 될 것이다. 라인 단위로 처리하기 위한 방법은 꼬리 재귀 방법을 이용하는 것이다. 기본 구조는 다음과 같다.

```
\def\comment#1^^M{... \comment}
```

위 매크로가 제대로 동작하려면, 우선 줄의 끝을 나타내는 ^^M의 카테고리 코드를 변경해야 한다.

```
\def\comment{\begingroup \catcode`\^^M=12 \xcomment}
{\catcode`\^^M=12 \endlinechar=-1 %
  \gdef\xcomment#1^^M{ ... \xcomment}}
```

\endlinechar의 값을 음수로 변경한 이유는 (기본값은 13) 위의 매크로 정의에서 매 라인의 끝마다 %를 넣어야 하는데 그것이 귀찮으므로, 그것을 안해도 되도록 하기 위함이니다. (TEX에서는 \endlinechar의 값이 음수이거나 255보다 크면, 모든 라인의 끝에 %를 삽입한 효과를 낸다.)

위의 매크로는 어느 시점에서는 끝나야한다. 그 방법은 꼬리 재귀 방법에서 자주 사용하던 방법을 사용하면 된다.

```
{\catcode`\^^M=12 \endlinechar=-1 %
  \gdef\xcomment#1^^M{\def\test{#1}
  \ifx\test\endcomment \let\next=\endgroup
  \else \let\next=\xcomment \fi
  \next}
}
\def\endcomment{\endcomment}
```

위에서 \endcomment는 실행되는 것이 아니라, 주석 처리 환경의 끝이라는 것을 나타낸다.

이것으로 TEX의 버퍼가 꽉차버리는 일은 없어졌다. 그렇다면, 앞서 언급한 outer 매크로나 짝이 맞지 않는 괄호를 사용하지 못한다는 단점은 어떤 방법으로 해결하면 될까? 이것은 아주 간단하다. 위의 \comment ... \endcomment를 하는 동안 모든 문자의 카테고리 코드를 12로 만들어 verbatim 모드로 바꾸어 주면 된다! 이를 위해서 plain TeX에는 아주 유용한 매크로 \dospecials이 다음과 같이 정의되어 있다.

```
\def\dospecials{\do\ \do\\\do\{\do\}\do$\do\&%
  \do\#\do\^\do\^^K\do\_\do\^^A\do\%\do\~}
```

이 \dospecials를 이용하면, \comment는 다음과 같다.

```
\def\makeinnocent#1{\catcode`#1=12 }
\def\comment{\begingroup
  \let\do=\makeinnocent \dospecials
  \endlinechar`\^^M \catcode`\^^M=12 \xcomment}
```

이것으로 인자로 어떠한 문자가 와도 해결이 된다. 이것으로 다 끝난 것일까? 문제가 이렇게 쉽게 해결 된다면, 얼마나 좋을까. (사실 그다지 쉽지도 않았지만.)

Verbatim 모드로 바꾸면 한 가지 문제가 발생한다. 그 문제는 공교롭게도 \comment ... \endcomment 환경에서 이 환경을 끝낼 때, 일어난다. 다시 자세히 살펴보면 우리는 \comment를 시작하면서 뒤에 나오는 모든

문자를 verbatim 환경으로 바꿔버렸기 때문에 이 환경을 끝내고자 하는 명령어 \endcomment 마저도 명령어로 보는 것이아니라 단순한 문자들로 보는 것이다. 이를 명령어로 인식하게 하려면 다음과 같이 하면 된다.

```
{\escapechar=-1
  \xdef\endcomment{\string\endcomment}}
```

한 가지 확실히 해둘 것은 위의 매크로를 단순히 아래와 같이 하면 안된다는 것이다.

```
\edef\endcomment{\string\endcomment}}
```

왜냐하면, 이와 같이 하면 '\' 뿐만 아니라 'endcomment' 마저도 카테고리 코드가 12가 되기 때문이다. 'endcomment'는 11이 되어야 한다.

참고문헌

[1] Donald E. Knuth, *The TEXbook* (Addison-Wesley, 1986).

[2] Victor Eijkhout, *TEX by Topic, , A TEXnician's Reference* (Addison-Wesley, 1992).

[3] Amy Hendrickson, *Getting TEXnical: Insights into TEX Macro Writing Techniques*, TUGboat, Volume 11 (1990), No. 3—1990 Conference Proceedings

2 초미니 한글 플레인 텍

김도현[3)]

이호석님의 은방울 트루타입에서 일부를 추출하여 type1 폰트를 하나 만들고, 이를 토대로 삼아 초미니 (폰트 포함해서 20Kb 남짓) 한글 plainTEX을 작성해 보았습니다.

실제로 사용할 일은 전혀 없을테지만, 심심해서...

세벌식 자판으로 직접 입력할 수 있는 문자만 지원합니다. 한자는 당근 안 되고, 특수문자 중에서는 가운데점, 참고표, 따옴표 정도만 지원합니다.

정말 심심해서 돌아가시겠다 싶으신 분만 받으세요...

file: jmoplain.tex

```
% utf8x.def는 UTF-8 인코딩 문자를 unicode point number로
% 바꾸어준다. e.g., ``가''는 UTF-8에서 ^^ea^^b0^^80으로
% 표현되는데 이를 44032("AC00)로 바꾸어 unichar의 인자로 넘겨주는
% 일을 utf8x.def가 담당한다.
\input utf8x.def
% lucenc.def는 그렇게 받은 unicode point number가 어느 subfont의
% 어느 char에 해당하는지 확인하여 그 char를 찍어준다.
% e.g., 44032는 당연히 "AC번째 폰트의 0번째 char이다.
% 따라서 \csname\LUCfont ac\endcsname, 즉 bangwlac라는 이름의 폰트(tfm)의
% 첫번째 글리프에 해당하는 박스를 그린다.
\input lucenc.def
% 하지만 jmoplain은, ucsplain과 달리,
% U+AC00의 완성된 음절 글리프를 이용하지 않고 모든 한글을
% U+1100 영역의 자소로 찍는 방법을 택한다. 폰트 갯수 단 한 개로 모든
% 한글을 표현해보자는 생각에서다. 따라서 bangwl11이라는
% 한 개의 폰트만으로 현대 한글 전 영역을 카바한다. 모양을 형편없지만...
\def\hfont#1#2{\def\LUCfont{#1}\def\LUCsize{#2}}
\def\hfontname#1{\def\LUCfont{#1}}\hfontname{bangwl}
\def\hfontsize#1{\def\LUCsize{#1}}\hfontsize{at 10pt}
% unichar의 파라미터는 unicode point number다. 이 중 한글에 해당하는 놈은
% 적절한 방법으로 조작하여 bangwl11로 찍고, 특수문자 중 세법식최종으로
% 입력가능한 놈은 tcmr1000 폰트에서 해당 모양을 찾아 찍는다. 그외는
% [U+AC00]의 모양을 출력하여 지원되지 않는 영역임을 분명히 한다.
\def\unichar#1{%
  % 세법식으로 직접 입력가능한 특수문자 몇개를 우선 처리한다.
  \ifnum #1="00B7 \tsonecmr{"B7}\else %· 가운데점은 tcmr1000의 "B7번째임.
  \ifnum #1=8251 \tsonecmr{'270}\else % ※ 참고표는 tcrm1000의 '270번째임.
  \ifnum #1=8220 ``\else % `` 여는 겹따옴표는 단순히 ``로 대치함.
  \ifnum #1=8221 ''\else % '' 닫는 겹따옴표는 단순히 ''로 대치함.
  \ifnum #1=65510 \tsonecmr{'216}\else % ₩ 원화기호는 tcmr1000의 '216번째임.
  % U+1100보다 앞에 나오는 글자들은 missingunichar로 넘겨 [U+AC00]형식을 식자.
  \ifnum #1<"1100 \missingunichar{#1}\else
```

3) http://www.ktug.or.kr/jsboard/read.php?table=contrib&no=3382

```
33    % U+1100 영역의 자모는 그대로 bangwl11을 이용해 식자한다.
34    \ifnum #1<"1200 \unihangulchar{#1}\else
35    % U+3131미만 글자들도 missingunichar로...
36    \ifnum #1<"3131 \missingunichar{#1}\else
37    % U+3131영역의 호환자모는 table방식으로 U+1100영역으로 전환시킨다.
38    \ifnum #1<"3164 \unihanguljamocompat{#1}\else
39    % U+AC00미만의 글자들(대부분 한자)도 missingunichar로...
40    \ifnum #1<"AC00 \missingunichar{#1}\else
41    % U+AC00 영역의 한글음절은 음절->자모 변환알고리즘을 이용하여
42    % U+1100영역으로 전환한다. 역시 종국적으로 bangwl11로 식자됨.
43    \ifnum #1<"D800 \unihangulsyllable{#1}\else
44    % 그 밖의 글자들도 missingunichar...
45    \missingunichar{#1}\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi }
46  % 몇몇 특수문자를 tcrm1000 폰트를 가지고 식자한다.
47  \def\tsonecmr#1{\begingroup
48    \font\myfont=tcrm1000 \LUCsize
49    \myfont\char#1\endgroup}
50  % 한글과 일부 기호 이외 글자는 [U+AC00] 형식으로 표시함.
51  \def\missingunichar#1{[U+\ucsdectohex{#1}]}
52  % unicode point number가 십진수로 넘어오는데 이를 16진수로 바꾸어준다.
53  % 비슷한 알고리즘이 ucs.sty에서 쓰이고 있다. 이를 loop를 이용해 단순화한 것임.
54  \def\ucsdectohex#1{% only <= U+FFFF
55    \begingroup
56    \count100="10000
57    \loop
58      \count@=#1
59      \divide\count@ by\count100
60      \multiply\count@ by-\count100
61      \advance\count@ by#1
62      \divide\count100 by "10
63      \divide\count@ by\count100
64      \ifcase\count@ 0\or 1\or 2\or 3\or 4\or 5\or 6\or 7\or
65                     8\or 9\or A\or B\or C\or D\or E\or F\fi
66      \ifnum\count100>1 \repeat
67    \endgroup }
68  % unihangulchar (여기서는 U+1100 한글 자모로 직접 입력된 것을 처리하게 됨)
69  \def\unihangulchar#1{%
70    % 우선 이전 글자가 금칙문자인지 확인해서 줄바꿈허용 명령을 삽입한다.
71    \breakbeforehangulchar{#1}%
72    % 아무 조작없이 lucenc.def가 정의하는 \lucchar에게 인자를 넘겨준다.
73    \begingroup\lucchar{#1}\endgroup
74    % 한글이 방금 식자되었음을 특정 spacefactor에 의하여 알린다.
75    \hangulspacefactor{#1}%
76    % 다음 토큰이 여는 괄호이면 break를 허용한다.
77    \expandafter\breakBeforeAsciiOParen }
78  % U+3131 호환자모는 U+1100 영역의 자모로 변환하여 식자한다.
79  \def\unihanguljamocompat#1{%
80    \breakbeforehangulchar{"1100}% 줄바꿈 허용.
81    \count@=#1
82    \ifnum\count@ > "314E
83      % 인자가 모음이면 그 순서 그대로 자모영역으로 대체할 수 있다.
```

```
    \advance\count@-"314F
    \advance\count@"1161
  \else
    % 자음이면 table 방식에 의하여 변환한다.
    \advance\count@-"3131
    \expandafter\count@\ifcase\count@
              "1100\or "1101\or "11AA\or "1102\or "11AC\or "11AD\or "1103\or
      "1104\or "1105\or "11B0\or "11B1\or "11B2\or "11B3\or "11B4\or "11B5\or
      "111A\or "1106\or "1107\or "1108\or "1121\or "1109\or "110A\or "110B\or
      "110C\or "110D\or "110E\or "110F\or "1110\or "1111\or "1112\fi
  \fi
  % U+1100 영역으로 전환되었으므로 width 1em hbox 속에 식자함.
  \hbox to 1em{\hss\expandafter\lucchar\expandafter{\the\count@}\hss}%
  \spacefactor=1001
  \expandafter\breakBeforeAsciiOParen }
% U+AC00 완성 음절은 음절->자모 변환알고리즘을 이용해서 U+1100영역으로.
% http://www.unicode.org/reports/tr15/#Hangul 참조.
\def\unihangulsyllable#1{%
  \count@=#1
  \advance\count@-"AC00
  \begingroup% leading consonant
    \divide\count@ 588
    \advance\count@ "1100
    \expandafter\breakbeforehangulchar\expandafter{\the\count@}%
    \expandafter\lucchar\expandafter{\the\count@}\endgroup
  \begingroup% vowel
    \count100=\count@
    \divide\count100 588
    \multiply\count100 588
    \advance\count@-\count100
    \divide\count@ 28
    \advance\count@ "1161
    \expandafter\lucchar\expandafter{\the\count@}\endgroup
  \begingroup% trailing consonant
    \count100=\count@
    \divide\count100 28
    \multiply\count100 28
    \advance\count@-\count100
    \ifnum\count@=0 \else
      \advance\count@ "11A7
      \expandafter\lucchar\expandafter{\the\count@}\fi\endgroup
  \spacefactor=1001
  \expandafter\breakBeforeAsciiOParen }
% 금칙문자 몇개
\sfcode`\(=998
\sfcode`\`=998
\sfcode`\[=998
% 한글 식자 전에 필요하면 줄바꿈을 허용.
\def\breakbeforehangulchar#1{%
  \ifhmode % horizontal mode에서만...
  \ifdim\lastskip=0pt % 직전에 glue가 없을 때만...
```

```
135     % 초성자음 앞에서만...
136     \ifnum #1>"10FF
137     \ifnum #1<"1160
138     % 직전 글자가 한글이면 discretionary(재량하이픈)를 삽입...
139     \ifnum\spacefactor=1001
140       \discretionary{}{}{}\else
141     % 금칙문자 뒤가 아니면 0pt짜리 glue 삽입...
142     \ifnum\spacefactor>998
143       \hskip0pt\relax
144     \fi\fi\fi\fi\fi\fi }
145   % 중성 또는 종성 다음에 한글이 식자되었음을 spacefactor에 의해 알림.
146   % hlatex도 비슷한 방법을 이용하여 줄바꿈 허용 여부를 결정한다.
147   \def\hangulspacefactor#1{%
148     \ifnum #1>"115F
149     \ifnum #1<"1200
150         \spacefactor=1001
151     \fi\fi }
152   % 다음 글자가 여는 괄호이면 0pt짜리 glue 삽입...
153   \def\breakBeforeAsciiOParen{\futurelet\nextglyph\breakbeforeasciioparen}
154   \def\breakbeforeasciioparen{\ifx (\nextglyph\hskip0pt\relax\fi}
155   % lucenc.def가 정의하는 luc@loadfont 재정의.
156   % 1. 원래 font size가 무시되고 있었는데 이를 무시하지 않게 하게 위해...
157   % 2. 한글 검색 추출을 위한 AddUniSubCmap를 삽입하기 위해...
158   \def\luc@loadfont{%
159     % \LUCfont=bangwl; \LUC@plane@LUC=11; \LUCsize=at 10pt(default)
160     % 이 폰트가 아직 정의되지 않았다면...
161     \expandafter\ifx
162         \csname LUC-font-\LUCfont\LUC@plane@LUC-\LUCsize\endcsname\relax
163       % 폰트를 전역적으로 정의함.
164       \global\expandafter
165         \font\csname LUC-font-\LUCfont\LUC@plane@LUC-\LUCsize\endcsname=%
166         \LUCfont\LUC@plane@LUC\space\LUCsize
167       % pdftex이 실행되고 있다면 당해 폰트에 해당하는 ToUnicode cmap 삽입.
168       \ifx\pdfoutput\undefined\else\ifx\pdfoutput\relax\else\ifnum\pdfoutput>0
169         \AddUniSubCmap{\LUCfont}{\LUC@plane@LUC}{\LUCsize}\fi\fi\fi
170     \fi
171     % 이 폰트를 사용하겠다고 선언함.
172     \csname LUC-font-\LUCfont\LUC@plane@LUC-\LUCsize\endcsname }
173   % pdftex이 지원하는 ToUnicode cmap을 삽입 루틴.
174   % http://www.tug.org/pipermail/pdftex/2001-May/000894.html 참조.
175   % pdftex 이외의 경우 당해 유틸의 능력에 의존. dvipdfmx는 알아서 잘 처리해준다.
176   \def\AddUniSubCmap#1#2#3{%
177     \immediate\pdfobj stream{%
178       /CIDInit /ProcSet findresource begin
179       12 dict begin
180       begincmap
181       /CIDSystemInfo << /Registry (TeX) /Ordering (Uni#2) /Supplement 0 >> def
182       /CMapName /TeX-Uni#2-0 def
183       1 begincodespacerange <00> <FF> endcodespacerange
184       1 beginbfrange <00> <FF> <#200> endbfrange
185       endcmap
```

```
186       CMapName currentdict /CMap defineresource pop end end
187     }\expandafter\pdffontattr\csname LUC-font-#1#2-#3\endcsname{/ToUnicode
188       \the\pdflastobj\space 0 R}}
189   \endinput
```

file:attached 

3 Memoir 배우기

Editorial: LaTeX을 사용하는 즐거움은 문서작성입니다. 아름답고 견고한 문서를 작성하는 도구 LaTeX의 활용 가능성을 극대화하는 클래스로서 memoir의 명성은 잘 알려져 있고, 한글을 사용하는 데도 큰 지장이 없을 정도가 되었습니다. 조인성 교수님의 글은 완성된 소개서가 아니라 문제를 던져주고 해결하도록 유도하는 "발제" 문서입니다. 이 문서의 내용을 익히고 부족하거나 질문만이 있는 부분에 답변하는 과정에서 memoir 문서 작성의 재미를 느낄 수 있을 것입니다.

조인성[4)]

이 문서에서는 memoir와 memhangul-ucs를 연습하는 과정을 담고자 한다. 그렇게 하는 첫째 이유는 memoir를 계속 사용하면서 나중에 필요할 때 개인적으로 참고하기 위해서이고 또 다른 이유는 나의 연습과정 기록이 memoir를 처음 배우고자 하는 사람에게 혹시 도움이 될 수 있을지도 모르겠다는 생각 때문이다. 개인적으로 또는 초보자들이 쉽게 사용하도록 하기 위해, 이 문서에서 표현된 모든 내용의 소스를 소스파일을 보지 않고도 결과 파일에서 직접 볼 수 있도록 했다. 이 문서는 latex 초보자를 위한 것이지만, latex을 처음 사용하는 사람들을 위한 것은 아니다. 이 문서는 latex으로 문서를 만들기 위해서는 `\documentclass`로 시작하고 `\begin{document}`와 `\end{document}` 사이에 본문을 타이핑해야 한다는 것 정도를 책을 보지 않고도 아는 사람을 대상 으로 한다. 연습과정의 거의 대부분은 memucs manual(김강수 역)을 따라 한 것에 지나지 않는다. 하지 만 몇 군데에서는 manual의 내용과 다르게 표현되기도 했는데, 이는 manual의 내용을 나름대로 이해해서 연습한 것을 적은 것이므로, **잘못 이해한 채로 적혀 있을 가능성을 배제할 수 없음**을 밝혀 둔다. 아울러, 전체 문서의 작성의 내용을 끌어가고 표현하는 데에 있어서도 일관성을 유 지하려는 **약간의** 노력을 기울였으나, 시간제약이 부담이 되거나 게으름이 날 때에는 이 노력을 과감히(?) 포기하였음도 밝혀 둔다. 또한 '초보자'의 연습기록이므로, 해당 기능이 memoir 고유 의 기능인지 아니면 표준 LATEX클래스에서도 작동하는 것인지를 일일이 구분해낼 수는 없었다는 치명적인 제약도 함께 작용했다는 것도 밝혀 둔다. **가장 부끄러운 것**은, 연습하기 위해 인덱스를 몇 개 작성했을 뿐, 진정한 인덱스를 만들지 못했다는 것이다. 이 문서는, 어떤 관점에서 보면, 초보자의 입장을 충실하게 유지하면서 작성된 문서일 수 밖에 없다.

이 문서는 이번 학기가 시작되기 전까지의 연습기록이고, 학기 시작 후로는 거의 볼 겨를이 없었다. 이 문서를 현재의 상태로 공개하는 것은 내가 원하는 바와는 매우 거리가 먼 일이기는 하나, '도은이 아빠'의 지속될 것같은 공개 독촉 압력에 굴복하기로 결정한 것은 내 책임이다.

`file:attached` 

4) `http://faq.ktug.or.kr/faq/MemhangulClass`

LaTeX초보의 memoir + memhangul-ucs 연습기록

조인성

2006년 4월 27일

요 약

이 문서에서는 memoir와 memhangul-ucs를 연습하는 과정을 담고자 한다. 그렇게 하는 첫째 이유는 memoir를 계속 사용하면서 나중에 필요할 때 개인적으로 참고하기 위해서이고 또 다른 이유는 나의 연습과정 기록이 memoir를 처음 배우고자 하는 사람에게 혹시 도움이 될 수 있을지도 모르겠다는 생각 때문이다. 개인적으로 또는 초보자들이 쉽게 사용하도록 하기 위해, 이 문서에서 표현된 모든 내용의 소스를 소스파일을 보지 않고도 결과 파일에서 직접 볼 수 있도록 했다.

이 문서는 latex 초보자를 위한 것이지만, latex을 처음 사용하는 사람들을 위한 것은 아니다. 이 문서는 latex으로 문서를 만들기 위해서는 `\documentclass`로 시작하고 `\begin{document}`와 `\end{document}` 사이에 본문을 타이핑해야 한다는 것 정도를 책을 보지 않고도 아는 사람을 대상으로 한다.

연습과정의 거의 대부분은 memucs manual(김강수 역)을 따라 한 것에 지나지 않는다. 하지만 몇 군데에서는 manual의 내용과 다르게 표현되기도 했는데, 이는 manual의 내용을 나름대로 이해해서 연습한 것을 적은 것이므로, 잘 못 이해한 채로 적혀 있을 가능성을 배제할 수 없음을 밝혀 둔다. 아울러, 전체 문서의 작성의 내용을 끌어가고 표현하는 데에 있어서도 일관성을 유지하려는 약간의 노력을 기울였으나, 시간제약이 부담이 되거나 게으름이 날 때에는 이 노력을 과감히(?) 포기하였음도 밝혀 둔다. 또한 '초보자'의 연습기록이므로, 해당 기능이 memoir 고유의 기능인지 아니면 표준 LaTeX클래스에서도 작동하는 것인지를 일일이 구분해낼 수는 없었다는 치명적인 제약도 함께 작용했다는 것도 밝혀 둔다. 가장 부끄러운 것은, 연습하기 위해 인덱스를 몇 개 작성했을 뿐, 진정한 인덱스를 만들지 못했다는 것이다. 이 문서는, 어떤 관점에서 보면, 초보자의 입장을 충실하게 유지하면서 작성된 문서일 수 밖에 없다.

여기는 초록(abstract)이 오는 자리이다. 요즘은 초록이라는 말 대신에 요약이라는 말을 많이 쓴다. 현재 이 문서에도 요약이 디폴트로 쓰였다. 특이한 점은 요약의 첫째 문단에서 indentation이 이루어지고 있다는 점이다.

이 문서는 이번 학기가 시작되기 전까지의 연습기록이고, 학기 시작 후로는 거의 볼 겨를이 없었다. 이 문서를 현재의 상태로 공개하는 것은 내가 원하는 바와는 매우 거리가 먼 일이기는 하나, '도은이 아빠'의 지속될 것같은 공개 독촉 압력에 굴복하기로 결정한 것은 내 책임이다.

차 례

그림 차례

표 차례

●가정: Windows XP에서 memoir + hlatex-ucs(dhucs) + memhangul-ucs를 설치했다고 하자.

이제, 다음 예제를 foo.tex로 저장하고 latex한다.

```
\documentclass{memoir}
\usepackage{memhangul-ucs}

\begin {document}
가장 간단한 파일
\end{document}
```

```
latex foo
```

MiKTEX에 포함된 dvi viewer인 Yap에서 inverse search나 forward search 기능을 이용하려면 --src-specials옵션을 넣어 latex한다. (어려운 일이 아니므로, 두 번 latex한다.)

```
latex --src-specials foo
```

dvipdfmx도 잘 된다.

역시 간단하지만, 몇 개의 패키지와 옵션을 추가한 파일을 보자.

```
\documentclass[a4paper]{memoir}
%\usepackage{amsmath,amssymb,graphicx}
%%\usepackage[hangul,nonfrench]{dhucs} % memhangul-ucs 안쓰고 한글 쓸때

%% Using memhangul-ucs
\usepackage[interworddefault,nonfrench,%
                  gremph,hangulpagestyle,%
                  adjustmath,pdfbookmark]{memhangul-ucs}
\xspaceskip=.7em plus .1em minus .1em %무시해도 됨
\adjustquotespacing %무시해도 됨

\begin {document}
조금 더 복잡하지만 역시 간단한 file.
\end{document}
```

\marginpar

boxedverbatim

\bvsides

\bvtopandbottom

\nobvbox

\linenumberfrequency

\resetbvlinenumber

\bvlinenumberinside

\bvlinenumberoutside

\bvbox

문서는 \frontmatter, \mainmatter, \backmatter로 구성한다. 위의 예에서 보듯이 꼭 그래야 하는 것은 아니지만 책과 같은 복잡한 문서에서는 이 기능들을 사용하는 것이 편리하다. 이 기능들은 페이지 번호 붙이기, 표제 번호 붙이기 등을 적절하게 조절하는 기능을 포함한다.

boxedverbatim

다음 예는 이러한 구성을 간략히 나타내고 있다.

```
\documentclass{memoir}
\usepackage{memhangul-ucs}

\begin {document}

\title{\bfseries \Huge memoir + memhangul-ucs 연습}
\author{\LaTeX 초보}
\date{\today} %날짜틀 생략하려면 \empty

\frontmatter %%%%%%%%%%
\maketitle % 위에서 작성한 타이틀을 출력한다.

\mainmatter %%%%%%%%%%
본문이 오는 곳이다.

\backmatter %%%%%%%%%%
참고문헌 등이 온다.
\end{document}
```

글을 쓰고 있는 여기는 mainmatter에 해당하는 곳이다. \mainmatter 선언의 기능은 페이지 번호를 아라비아 숫자로 1부터 시작하도록 하며, 장절명령에 번호를 붙이도록 하는 것 등을 포함한다. 만일 페이지 번호매김과 그 형식에 변화를 주지 않고 지금까지의 형식을 유지하고 싶으면 별표를 붙여 \mainmatter*로 선언하면 된다.

\fancybreak

\plainbreak

* * *

아직은 part, chapter, section 등을 시작하지 않은 상태이다. 이제, chapter에 관해 살펴보기로 하자.

제 1 장

단순 챕터 시작하기

이 장은 \chapter{단순 챕터 시작하기}와 같이 하여 시작했다. 여기는 chapter와 section 사이이다. 즉 chapter는 시작했지만 아직 section이 시작되지 않았다. 이제 절을 보기로 하자.

1.1 절은 어떻게 생겼나? (default: 절까지만 번호 매김)

이 절에서는 장절의 번호매김이 디폴트로 되어 있다. 디폴트는 section까지만 번호를 매기는 것이다. 즉, 표준클래스에서 다음과 같이 한 것과 같다.

```
\setcounter{secnumdepth}{2}
```
(∗)[1]

subsection은 어떻게? (default: 소절 이하에는 번호가 붙지 않음)

이 subsection에는 번호가 붙지 않았다.

subsubsection은 어떻게?

이 subsubsection에서도 번호가 붙지 않았다.

[1] (∗)표시는 이 표현이 (memoir에서만이 아니라) LaTeX표준클래스에서도 유효하다는 것을 나타내기 위해 사용되었다. Observe that although we have a footnote before this page, the footnote number starts from one again because this belongs to a new chapter.

1.2 또 다른 절은 어떻게 생겼나? (장절 번호매기기...)

이 절에서는 장절의 번호매김을 subsubsection까지 붙이기로 한다. 이를 위해서는 다음과 같이 하면 된다.

```
\maxsecnumdepth{subsubsection}
```

컴파일 중에 `\mainmatter` 명령이 읽히면, secnumdepth가 `\maxsecnumdepth`로 설정된다. `\maxsecnumdepth` 대신 `\setsecnumdepth`를 써도 된다. 이들 명령은 preamble을 포함하여, 어느 곳이든, 번호를 붙이거나 변경하고자 하는 (sub)section 이전에만 오면 된다.

1.2.1 subsection에 번호 붙었나?

`\verbfootnote` 이 subsection에는 번호가 붙었다.[2]

1.2.1.1 subsubsection에 번호 붙었나?

이 subsubsection에도 번호가 붙었다. 이 이후로 계속 subsubsection까지 번호붙이기를 하기로 하자. 맨 마지막에 사용된 `\maxsecnumdepth` 또는 `\setsecnumdepth`가 (다시 바꿔기 전까지) 계속 유효하다.

1.3 Chapter Style 이용하기

1.3.1 Chapter 시작하기

chapter 시작하기는 `\openright`가 디폴트이며 `\opneleft`와 `\openany`도 사용할 수 있다.

이들은 문서클래스의 옵션으로 설정할 수도 있으나, 문서 내 어디에서든 선언하여 사용할 수도 있다. 예를 들면 다음과 같다.

```
\opneany
\chapter{왼쪽 오른쪽 아무데서나 열리는 챕터}
. . .
```

[2]특정 장절에 번호를 붙이지 않으려면 `\chapter*{...}` 또는 `\subsubsection*{...}`와 같이 장절명령 뒤에 *를 붙이면 된다. 이 각주 작성은 `\verbfootnote{특정 장절에...}`와 같이 하여 작성되었는데, 이는 표준적인 `\footnote{...}`에서와는 달리 각주의 text 안에서 `\verb+...+`를 쓸 수 있게 한다.

`\cleartorecto`와 `\cleartoverso` 등을 포함한 페이지 이동 명령을 김강수(2006) 『LaTeX-memoir로 책 만들기』 p.43 에서 보다 상세히 다루고 있다.

제 2 장

companion 챕터 스타일

여기에서는 장절표제형식의 변형에 대해 연습한다. 기본형식(default style)을 이용하면 되므로 시간이 없으면 나중에 연습해도 된다.

* * *

이 장은 다음과 같이 시작하였다.

```
%\chapterstyle{default}       % Why is this used here?
\chapterstyle{companion}
\chapter[차례 companion][면주 companion style]{companion 챕터 스타일}

여기에서는 장절표제형식의 변형에 ...
```

2.1 차례와 면주의 장절표제

`\chapter` 명령은 간단하게는 `\chapter{`*text*`}`의 형태로 쓰인다. 이 때 입력된 *text*의 내용이 table of contents와 (짝수쪽) heading에서 사용된다. 만약 `\chapter[`*option*`]{`*text*`}`와 같이 하면, 차례와 면주에 *option*의 내용이 나타난다.

또한 `\chapter[`*option 1*`][`*option 2*`]{`*text*`}`와 같이 하면, 차례에는 *option 1*, 면주에는 *option 2*의 내용이 나타난다. 실제로, 이 쪽이나 다음의 한두 쪽을 살펴보면 heading에서 '면주 COMPANION STYLE'을 볼 수 있을 것이다.

2.2 chapter style

장절표제의 형태를 바꾸려면 default나 companion 등과 같이 미리 정의된 chapterstyle을 이용하거나 새로운 chapterstyle을 정의하여 사용할 수 있다.

이 전 장은 `\chapterstyle{companion}`를 이용하였고, 이 장에서는 default chapterstyle을 이용하였다. companion chapterstyle은 *The LaTeX Companion*의 chapterstyle을 흉내낸 것이라고 한다.

2.3 Some More Variations: Chapter

(to be completed. . .)

3 드디어 본문 조판하기

이 장은 section 장절 스타일로 조판되었다.

3.1 문단모양

표준 문단의 모양을 제어하는 두 개의 기본적인 매개변수 중 `\parindent`는 문단 첫줄의 들여쓰기 값이고, `\parskip`은 문단과 문단 사이의 수직 간격이다.

이 클래스의 디폴트 `\parindent`는 약 1.52em이며, `\parskip` 값은 일반적으로 0pt이다.

<START—START로 시작하는 이 문단과 END로 끝나는 다음 문단, 두 문단에만 `\parindent` 값이 5em, `\parskip` 값이 10pt가 적용되도록 이 문단 중간에

`\setlength{\parindent}{5em}`와

`\setlength{\parskip}{10pt}`를

선언하여 해당 값을 조정하였다.

<START에서 END>까지 두 문단을 괄호 {. . . }로 묶었으므로, 위에서 정한 값은 END 이후 바로 다음으로 이어지는 문단부터는 적용되지 않고, 원래 디폴트로 정해졌던 값이 적용된다. END>

만약 `\parindent`가 음수(−) 값을 갖는다면 문단의 첫번째 줄은 왼쪽 여백을 침범하여 '내어밀기'될 것이다.

예로, 이 문단은 `\setlength{\parindent}{-5em}`를 사용하여 여백쪽으로 5em만큼 내어밀기가 되도록 식자되었다. 이 선언은 계속 유효하므로 이 선언에 영향을 받는 범위를 지정해주도록 한다. 간단하게는 해당부분을 괄호 {. . . }로 묶어주면 된다.

3.1.1 여러 줄 들여밀기(?) 내어밀기(?)

여러 줄 들여/내어밀기는 한 문단의 처음이나 나중의 몇 줄이 문단 내 나머지 행의 길이와 다르게 하는 것이다. 표준 indentation은 처음 한 줄의 길이만 다른 special case로

이해할 수 있다.

3.1.1.1 hangpara

여러 줄 들여쓰기는 \hangpara{*indent*}{*num*}을 선언함으로써 실행된다. 첫째 인자 *indent*는 들여밀기 하는 길이, 둘째 인자는 *num*는 행의 수를 의미한다. *indent*와 *num*이 양(+)수이냐 음(−)수이냐에 따라 효과가 다르다. 예를 들면, \hangpara{10mm}{-3}은 문단의 처음 세행을 오른쪽으로 10mm만큼 '들여밀기' 하라는 것을 의미한다. 두 인자와 그 부호의 네 가지 조합을 정리하면 다음 표와 같다.

center

tabular

	num(*N*)	*indent*
(+)	처음 *N*행 제외하고	(행의 처음을) 오른쪽으로 들여밀기
(−)	처음 *N*행을	(행의 끝을) 왼쪽으로 들여밀기

\textvisiblespace

이 선언은 문단 내의 어느 곳에 위치해도 된다. 단, 문단의 첫 머리에 올 때에 주의할 점은 \hangpara 명령과 문단의 첫 단어 사이에 공백(␣)이 없어야 한다는 것이다. 이유는 그 ␣가 의도하지 않은 indentation효과를 가져오기 때문이다. \hangpara가 적용된 문단은 문단의 첫 부분에 통상적인 indentation이 되지 않는다.

다음에서 네가지 경우의 예를 보자.

* * *

0이 문단은 \hangpara{3em}{2}으로 작성되었다. 이 선언은 '처음 두행을 제외하고 오른쪽으로 3em만큼 들여밀라'는 것을 의미한다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다.

* * *

0이 문단은 \hangpara{3em}{-2}으로 작성되었다. 이 선언은 '처음 두행을 오른쪽으로 3em만큼 들여밀라'는 것을 의미한다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다.

* * *

0이 문단은 \hangpara{-3em}{2}으로 작성되었다. 이 선언은 '처음 두행을 제외하고 왼쪽으로 3em만큼 들여밀라'는 것을 의미한다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다.

* * *

0이 문단은 \hangpara{-3em}{-2}으로 작성되었다. 이 선언은 '처음 두행을 왼쪽으로 3em만큼 들여밀라'는 것을 의미한다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다. 여러 줄 들여밀기 연습문장입니다.

3.1.1.2 hangparas 환경

hangparas 환경은 환경내의 모든 문단에 \hangpara의 기능이 적용되게 한다. 이를 위해서는 \begin{hangparas}{*indent*}{*num*} ... \end{hangparas}와 같이 한다.

3.1.2 hangfrom: 간단한 내어/들여(?) 밀기 문단

간단한 내어밀기 문단은 \hangfrom*text* 명령을 사용하면 된다.

* * *

이 문단은 간단한 내어밀기 문단의 예를 보인 것이다. 이 문단과 같은 효과를 내려면 문단을 \hangfrom{이 문단은 간단한 내어밀기... 와 같이 시작하면 된다. 간단하다!

3.2 문단폭 바꾸기: adjustwidth 환경

3.2.1 adjustwidth

문단 폭을 일시적으로 바꿀 때는 adjustwidth 환경을 사용하면 편리하다. 이 환경은 다음과 같이 사용한다.

```
\begin{adjustwidth}{<left>}{<right>} ... \end{adjustwidth}
```

adjustwidth환경은 환경내 문단에 *left*길이를 왼쪽 마진에 *right* 길이를 오른쪽 마진에 더해주는 기능을 한다. 이 값이 양(+)수이면 문단폭이 줄어들고 음(−)수이면 늘어난다. 왼쪽과 오른쪽을 2em만큼 들여쓰게 하면 quotation환경과 거의 같은 효과를 낼 수 있다.

3.2.2 adjustwidth*

별표붙은 adjustwidth*환경은 *left*를 등(spine)으로, *right*를 배(foredge)로 인식한다. 따라서 홀수쪽에서 두 환경의 효과는 같지만, 짝수쪽에서는 별표(*)가 붙으면 left 길이만큼 등(오른)쪽을 right 길이만큼 배(왼)쪽을 들여쓴다.[1]

* * *

예1: 이 문단은 \begin{adjustwidth}{5em}{2em}로 시작하여 문단의 왼쪽을 5em, 오른쪽을 2em만큼 들여쓰도록 지정하고, 지금 읽고 있는 문단의 내용을 쓴 후 \end{adjustwidth}로 끝나게 식자되었다.

예2: 이 문단은 \begin{adjustwidth}{2em}{5em}로 시작하여 문단의 왼쪽을 2em, 오른쪽을 5em만큼 들여쓰도록 지정하고, 지금 읽고 있는 문단의 내용을 쓴 후 \end{adjustwidth}로 끝나게 식자되었다.

예3: 이 문단은 \begin{adjustwidth*}{5em}{2em}로 시작하여 문단의 등쪽을 5em, 배쪽을 2em만큼 들여쓰도록 지정하고, 지금 읽고 있는 문단의 내용을 쓴 후 \end{adjustwidth*}로 끝나게 식자되었다.

* * *

위에 세 문단을 예로 들었다. 별표붙은 환경을 사용한 예3은 어느 페이지에 인쇄되느냐에 따라 그 결과가 다르다. 지금 현재 이 페이지가 홀수이면 예1과 같을 것이고, 짝수이면 예2와 같게 인쇄되었을 것이다. (Am I correct?)

3.2.3 physical paper의 중앙에 위치시키기

문단이나 제목 등을 판면(page)의 중앙이 아니라 인쇄용지의 중앙에 놓이게 하려면 다음과 같이 한다. \calccentering 명령과 adjustwidth* 환경을 이용하여 간단하게 판면을 용지 중앙에 오게 할 수 있다.[2]

\calccentering

\normalfont

이 문단은 편집영역(판면)이 아니라 용지의 중앙에 오도록 식자되었다. 이를 위해서는 기존에 정해진 임의의 길이—여기서는 \unitlength—를 \calccentering으로 측정하고 이를 등쪽에 더하고 배쪽에 더해준다.

```
\calccentering{\unitlength}
\begin{adjustwidth*}{\unitlength}{-\unitlength}
...
\end{adjustwidth*}
```

[1] adjustwidth환경내에서는 \noindent 기능이 꺼지는 것으로 보인다.

[2] This technique does not seem to be robust.

3.3 나열 문단: itemize, enumerate

3.3.1 description: 표준클래스 그대로

```
\begin{description}
  \item[description] 표준 \LaTeX에서와 같이 memoir에서도 ...
  \item[itemize] ...
  \item[enumerate] ...
\end{description}
```

description 표준 LaTeX에서와 같이 memoir에서도 description환경을 제공한다. 표준 LaTeX에서와 같이 memoir에서도 description환경을 제공한다.

itemize 환경 itemize에서는 확장된 기능을 제공한다. 환경 itemize에서는 확장된 기능을 제공한다. 환경 itemize에서는 확장된 기능을 제공한다.

enumerate 환경 enumerate에서도 확장된 기능을 제공한다. 환경 enumerate에서도 확장된 기능을 제공한다. 환경 enumerate에서도 확장된 기능을 제공한다.

3.3.2 확장된 itemize

```
\begin{itemize}[<mark>] \item ...  \end{itemize}
```

*mark*의 디폴트값은 •\textbullet이고 그 밖에 어떤 것도 올 수 있다. 예로, \tightlist

- – \textendash,
- ∗ \textasteriskcentered,
- → \rightarrow,
- 또는 ¶ \P(pilcrow) 등을 들 수 있다.

위의 예는 다음과 같이 하여 작성되었다.

```
\begin{itemize}[\bfseries\textperiodcentered]\tightlist
  \item \textendash\, \verb+\textendash+,
  \item \textreferencemark\, \verb+\textreferencemark+
  ...
\end{itemize}
```

\tightlist
\firmlist
\tightlists
\defaultlists

위의 예에서, 환경시작 직후에 사용된 \tightlist는 환경내의 나열항목 사이의 수직간격을 없애주는 역할을 한다. \firmlist는 조금 줄어든 수직간격을 얻게 한다.

어떤 때는 chapter 전체에서 매환경마다 \tightlist를 넣어야 할 필요가 있을 수도 있다. 이 때에는 chapter 초기에 (또는 환경시작 전에) \tightlists를 선언하면, 이 선언 이후의 모든 환경내에 \tightlist를 넣은 것과 같은 결과를 얻을 수 있다. 항목 사이의 간격을 표준대로 넓게 하고 싶으면 \defaultlists를 선언하면 된다.

3.3.3 확장된 enumerate

```
\begin{enumerate}[<style>] \item ...  \end{enumerate}
```

*style*의 디폴트값은 '1.'(즉, 아라비아숫자 + period)이다. 아라비아 숫자 1은 첫째 항목의 번호이며 그 후의 항목번호는 같은 스타일을 유지하면서 자연스럽게 증가된다. *style*은 1), [A], a., I. (i)등 여러가지 형태로 나타낼 수 있다.

*style*옵션의 값에 숫자와 알파벳이 아닌 문자(예로, •\textbullet)가 오면 itemize 환경의 글머리부호 *mark*를 •으로 지정한 것과 같은 효과를 가져온다.

*style*옵션에 한글 '가.'를 넣어도 문자로 간주되어 모든 항목의 번호가 '가. 가. ...' 처럼 표시된다. 항목 번호를 '가. 나. 다. ...'와 같이 표시하려면 다음과 같이 한다.[3]

가. 이 예에서는 [*style*]옵션 대신에 \renewcommand{\labelenumi}{\gana{enumi}.} 를 사용하였다.
나. \gana 대신에 \ogana를 사용하면 ㉮ ㉯ ㉰...와 같이 된다.
다. \pjaso를 사용하면 ㉠ ㉡ ㉢...과 같이 되고
라. \onum을 사용하면 ① ② ③...과 같이 된다.

[3] 이 방법은 LaTeX의 표준클래스에서 enumerate환경의 numbering style을 바꿀 때 쓰는 방법이다. \labelenumi 대신에 \labelitemi를 쓰면 itemize 환경의 글머리부호도 바꿀 수 있으나, memoir클래스에서 글머리부호 옵션인 *mark*옵션을 사용하는 것이 보다 편리하다 하겠다.

3.3.4 새로운 list 환경의 정의 (생략가능)

바쁜 사람들은 이 subsection을 생략하고 다음으로 넘어가도 된다.

3.3.4.1 새로운 환경

```
\begin{list}{<default-label>}{<code>} \item ...  \end{list}
```

예를 통해 보기로 하자.

```
%%%% 항제목을 이탤릭체로 하고 항목간 수직간격을 줄인 description 환경
\newcommand{\itlabel}[1]{\hspace\labelsep\normalfont\itshape #1}
\newenvironment{itdesc}%
  {\list{}{%
    \setlength{\labelsep}{0.5em}
    \setlength{\itemindent}{0pt}
    \setlength{\leftmargin}{\parindent}
    \setlength{\labelwidth}{\leftmargin}
    \addtolength{\labelwidth}{-\labelsep}
    \setlength{\listparindent}{\parindent}
    \setlength{\parsep}{\parskip}
    \setlength{\itemsep}{0.3\onelineskip} %항목간 수직간격 조절
    \let\makelabel\itlabel}}%
    {\endlist}
%%% 사용하는 것은 보통의 description환경과 같다.
\begin{itdesc}
  \item[italic]
  ...
\end{itdesc}
```

italic This item *label* is printed in italic shape. 이탤릭체로 하는 것이 표준적인 description환경의 표제보다 보기 좋은가?

description 전 항과 이 항의 항목간 간격은 tighter해졌나?

```
%% glossary list environment 정의하기
%% 다음과 같이 본문 또는 preamble에 정의한다.
\newenvironment{aglossary}%
```

```
   {\begin{list}{}% empty label
      {\setlength{\topsep}{\baselineskip}
       \setlength{\partopsep}{0pt}
       \setlength{\itemsep}{0.5\baselineskip}
       \setlength{\parsep}{0pt}
       \setlength{\leftmargin}{2em}
       \setlength{\rightmargin}{0em}
       \setlength{\listparindent}{1em}
       \setlength{\itemindent}{0em}
       \setlength{\labelwidth}{0em}
       \setlength{\labelsep}{2em}}}%
   {\end{list}}
\newcommand{\gloss}[1]{\item[\bfseries #1]\mbox{}\nopagebreak}

%% aglossary 환경 사용하기
\begin{aglossary}
  \gloss[function] A function from ...
  \gloss[한계대체율] ...
\end{aglossary}
```

위의 coding의 예를 따라 다음 나열문단을 작성하였다.

function A function from X to Y is a rule that assigns to each element in X one and only one element in Y.

한계대체율 한계대체율이란 동일한 만족수준을 유지하면서 X재 한 단위를 더 소비하기 위해 포기해야 하는 Y재의 양을 말한다.

3.3.4.2 환경 전후의 수직간격 조정

`center` 등의 trivlist 환경들을 사용하면 전후에 약간의 수직간격이 더해진다. 이 간격을 없애려면 환경시작 (바로) 전에 `\zerotrivseps`를 선언하라.

> 이 문단은 *center*환경을 이용하여 작성였다. 이 문단 전후에 추가되는 수직간격을 없애기 위해 환경시작 전에 `\zerotrivseps`를 선언했는데 간격이 없어졌나?

환경내에서 선언하면 원하는 결과를 얻을 수 없다. 환경 전후의 추가되는 수직간격을 원래대로 되돌리려면 `\restoretrivseps`를 선언하면 된다.

> 이 문단은 *center*환경을 이용하여 작성하였다. 이 문단 전후에 추가되는 수직간격을 회복하려고 환경시작 전에 `\restoretrivseps`를 선언했는데 간격이 돌아왔나?

간격이 (원래대로) 회복되었나? `\zerotrivseps`를 선언하는 것은 다음에서 의미하는 것과 같은 효과를 얻는다.

```
\renewcommand{\zerotrivseps}{%
\setlength{\topsep}{0pt}%
\setlength{\partopsep}{0pt}}
```

필요하다면 0pt 부분을 조정하여 fine tuning을 할 수 있을 것이다.

4 frontmatter(ToC etc.) and backmatter(BiB, INDEX)

이 장은 hangnum 챕터스타일로 조판되었다. `\hangsecnum` 을 선언하면 그 선언 이후의 절번호는 여백으로 빠져나간다.

4.1 ToC, LoF, and LoT

4.1.1 ToC만들기

`memoir` 클래스는 `\tableofcontents` 나 `\tableofcontents*`가 선언된 위치에서 ToC를 만든다. 일반적으로 `\tableofcontents`를 선언하는 곳은 `\mainmatter`를 선언하기 직전이면 된다. 하지만 바로 이곳에 `\tableofcontents`를 선언하면 바로 이곳에 '차례'가 만들어진다. (Try it yourself.)

YOU WANT ToC RIGHT HERE?

마찬가지로 `\listoffigures` 나 `\listoffigures*`가 선언된 위치에서 LoF가 만들어지고, `\listoftables` 나 `\listoftables*`가 선언된 위치에서 LoT가 만들어진다. 별표(∗) 붙은 선언은 자신의 heading을 ToC에 추가하지 않는다.

표준 LaTeX클래스에서와는 달리 `memoir` 클래스에서는 ToC, LoF, LoT를 새로운 페이지에서 시작하지 않는다. 새로운 페이지에서 시작하고 싶으면 `\cleardoublepage`나 `\clearpage`를 이용하면 된다. `\tableofcontents`를 선언하기 전에 `\chapterstyle{...}` 명령을 선언하여 챕터스타일을 조절할 수도 있다.

어떤 장절까지 ToC 엔트리에 넣을까 하는 것은 `\maxtocdepth{...}`를 사용하거나 `\settocdepth{...}`를 사용하여 조절한다. 기본값은 `\maxtocdepth{section}`이다. 이 명령은 preamble을 포함하여 문서 내 어느 곳에 와도 된다.

```
...
\maketitle
...
\cleardoublepage
\chapterstyle{demo}

\maxtocdepth{subsection}  %생략하면 section까지 표시
\tableofcontents*
\clearpage
\listoffigures
...
\mainmatter %%%%%%%%%%%
```

4.1.2 표제모양 제어하기

초보자가 표제모양을 제어할 필요가 있을까? 표제모양 제어의 필요성은 초보자인지의 여부와는 관계없는 문제이다. 문제는 초보자가 표제모양 제어 방법을 배울 필요가 있는 것인가이다. 지금 당장 필요하지 않으면 다음으로 미루는 것도 좋을 것이다.

본질적으로 ToC, LoF, LoT 표제는 chapter표제와 동일한 형식을 가지므로 현재의 chapterstyle에 의해 모양이 결정된다. 이 모양을 바꾸는 예를 몇 개 들기로 하자. `toc`를 `lof`나 `lot`로 바꾸어도 같은 논리가 성립한다.

* 표제가 오른쪽 정렬되어 Large italic 폰트로 식자되게 하려면:
  `\renewcommand{\printtoctitle}[1]{\hfill\Large\itshape #1}`
* 표제가 Large bold로 중앙정렬되기를 원하면:
  `\renewcommand{\printtoctitle}[1]{\centering\Huge\bfseries #1}`
* ToC의 첫 페이지는 empty 페이지 스타일이 되게 하려면:
  `\renewcommand{\aftertoctitle}{\thispagestyle{empty}\afterchaptertitle}`
* 'Page'라는 단어를 표제가 앉혀진 줄에 flushright되게 하려면:
  ```
  \renewcommand{\aftertoctitle}{%
     \par\nobreak \mbox{}\hfill{\normalfont Page}\par \nobreak}
  ```

`\printtoctitle`은 실제로 표제를 식자한다. (`\printchaptertitle`이 기본으로 설정되어 있다.) `\aftertoctitle`은 표제가 식자된 후에 불린다. (`\afterchaptertitle`이 기본으로 설정되어 있다.)

4.1.3 엔트리 식자

ToC 엔트리와 페이지를 연결하는 줄은 기본적으로 `\renewcommand{\cftdot}{.}`로 설정되어 있다. `\renewcommand{\cftdot}{\ensuremath{\ast}}`로 하면 점선이 아니라 $*$로 줄을 긋는다.

이 이외에 많은 조절기능이 있으나, 나중에 살펴보기로 하자.

4.2 Bibliography

4.2.1 thebibliography 환경

참고문헌 목록 또는 서지(bibliography)를 만드는 명령은 표준 LaTeX클래스의 것과 동일하지만 memoir 클래스에서 그 기능이 개선되었다. 색인(index)의 경우도 LaTeX의 명령과 동일하지만 그 기능이 현저히 개선되었다. bibliography와 index는 `\backmatter` 후에 온다.

```
\begin{thebibliograph}{<exlabel>}
  \bibitem [<x>]{StewartCal} Stewart, James, \textit{Calculus}, ...
  \bibitem ...
\end{thebibliography}
\bibname
```

bibliography는 `thebibliography`환경으로 만든다. 이 환경이 취하는 인자 *exlabel*은 bibliography에서 가장 넓은 label의 폭을 나타내기 위해서 주는 문자이다.[1] `\bibname`의 값이 표제로 사용되는데, 기본값은 'Bibliography'이다. `memhangul-ucs`에서 `\bibname`의 기본값은 '참고 문헌'이다. `\renewcommand{\bibname}{參考文獻}`을 `bibliography` 환경 전에 식자하면 표제를 '참고문헌'으로 바꿀 수 있다.

```
\renewcommand\prebibhook{...}
\renewcommand\postbibhook{...}

\setbiblabel{\textperiodcentered}
```

[1] 통상적으로 참고문헌이 9개 이하이면 *exlabel*을 문자 '9'로 주고, 99개 이하이면 문자 '99'로 준다. 이 경우 문자 '99'가 인자되는 폭이 label의 최대폭이 되는 것이다. 문자 'ii'의 폭은 더 좁을 것이고 문자 'MM'의 폭은 넓을 것이다. `\bibitem`의 두 인자 `[...]`와 `{...}`는 LaTeX클래스에서와 같은 의미를 가지므로 무슨 뜻인지를 적절한 문헌을 통해 알아볼 것을 권장한다.

```
\renewcommand{\bibname}{參考文獻}
\begin{thebibliograph}{99}
  \bibitem{HPW} Haeussler..., \textit{Introductory...}, ...
  \bibitem{Sydsaeter} Sydsaeter..., \textit{Essential...}, ...
  \bibitem ...
\end{thebibliography}
```

\prebibhook{...}과 \postbibhook{...}을 bibliography 환경 전에 두면 {...}의 내용을 각각 문헌목록의 첫 항목 전과 마지막 항목 후에 식자한다. \nobibintoc 선언은 ToC에 표제 'Bibliography'가 나타나지 않게 한다. 기본값은 표제가 나타나도록 하는 \bibintoc로 설정되어 있다.

위와 같이 식자하면 참고문헌은 다음과 같이 (유사하게) 나타난다.

[1] Haeussler, Ernest F., Richard S. Paul, and Richard Wood, *Introductory Mathematical Analysis*, 11th ed., 2003.

[2] Hughes-Hallett, Deborah, Andrew M. Gleason, and William G. McCallum, *Calculus: Single and Multivariable*, 4th ed., 2005.

[3] .....

참고문헌 항목간의 수직 간격은 \bibitemsep으로 조절할 수 있다. \bibitemsep의 기본값은 \itemsep의 값으로 지정되어 있다. 참고문헌은 list로 식자되므로, 항목간의 간격은 \bibitemsep + \parsep이다. \setlength{\bibitemsep}{-\parsep}을 하면 항목간 추가 간격을 제거할 수 있다. 그 결과는 다음과 같다.

[1] Haeussler, Ernest F., Richard S. Paul, and Richard Wood, *Introductory Mathematical Analysis*, 11th ed., 2003.

[2] Hughes-Hallett, Deborah, Andrew M. Gleason, and William G. McCallum, *Calculus: Single and Multivariable*, 4th ed., 2005.

[3] .....

4.2.2 내게 맞는 bibliography 만들기

다음과 같이, 번호 없이 들여쓰기만 되는 형태를 만들려면 어떻게 해야 하나?

Haeussler, Ernest F., Richard S. Paul, and Richard Wood, *Introductory Mathematical Analysis*, 11th ed., 2003.

Hughes-Hallett, Deborah, Andrew M. Gleason, and William G. McCallum, *Calculus: Single and Multivariable*, 4th ed., 2005.

.....

한 가지 방법으로 다음과 같이 fine-tuning하는 방법을 생각해 볼 수는 있겠다. 먼저 새로운 길이변수 \mybibindent를 정의하여 들여쓰기의 길이를 정의하는 것이 편리하다. 다음으로 번호를 없애는 것은 \setbiblabel{} 또는 \setbiblabel{\empty}을 명령하면 된다. 다음으로 \biblistextra를 재정의하여 원하는 문헌목록 형태를 위와 유사하게 만들 수 있다. \biblistextra의 기본값은 아무 것도 하지 않는 것이지만, 재정의를 통해 list 파라미터를 수정하는 데 사용된다.

단, \biblistextra를 이용하면 \setlength{\bibitemsep}{-\parsep}이 작동하지 않는다(I don't know why). 따라서 이를 위한 추가적인 조정이 필요하다. 여기서는 인위적으로 \vspace{-\parsep}를 넣어 조정했다. 이 두가지 일을 한꺼번에 하기 위해 임시로 새로운 명령 \mybibstyle을 정의하여 썼다.[2]

```
\newlength\mybibindent
\setlength\mybibindent{1.5em}

\renewcommand{\biblistextra}{%
  \setlength{\leftmargin}{\mybibindent}%
  \setlength{\itemindent}{0pt}%
}

\newcommand\mybibstyle{\vspace{-\parsep}\hspace{-\mybibindent}} %%%%%
\setlength{\bibitemsep}{-\parsep} %does not work with "\biblistextra"
\setbiblabel{}
\begin{thebibliography}{}
  \bibitem{HPW}\hspace{-\mybibindent}Haeussler, ......
  \bibitem ......
\end{thebibliography}
```

[2] \mybibindent와 \mybibstyle 명령은 사용자가 임시 방편 목적으로 정의하여 쓴 것이다.

정리 makeindex.exe 정리!정점정리 정리!재고정리

4.3 Index

4.3.1 index 만들기

이 클래스는 makeidx, showindex, index 패키지에서 제공하는 기능을 구현하므로, 이 패키지들과 함께 쓰면 안된다.

단어 '정리'를 인덱스에 포함시키려고 한다면 어느 곳에나 \index{정리}와 같이 명령하면 이 단어가 본문에는 나타나지 않고 인덱스 엔트리로 나타난다.

\makeindex를 preamble에 선언하면 index의 input파일인 foo.idx가 만들어진다. 이 foo.idx를 index의 output 파일인 foo.ind로 바꾸어주는 것은 makeindex.exe이다. (보통은 bibliography 이후에) \printindex를 선언하여 output 파일 foo.ind를 읽어 들임으로써 index를 만든다.

따라서, index를 포함한 결과를 얻으려면 다음과 같은 순서로 컴파일해야 한다.[3]

```
latex foo.tex   %\makeindex 선언으로 idx파일 만듦
makeindex foo.idx   %idx파일을 ind파일로 바꿈
latex --src-specials foo.tex   %\printindex 선언으로 ind파일을 읽음
```

인덱스 표제는 \indexname의 값이다. memoir에서 초기값은 'Index'이며 memhangul-ucs에서의 초기값은 '찾아보기'이다. 만약 어떤 이유에서든 인덱스의 표제를 '찾아서 보기'로 바꾸고 싶다면 \renewcommand\indexname{찾아서 보기}와 같이 하면된다.

4.3.2 index 만들기: Some tricks

'정리'라는 큰 엔트리에 속하는 작은 엔트리로 정점정리와 재고정리를 넣고 싶으면 다음과 같이 한다.

```
'정리'라는 큰 엔트리에 속하는 작은 엔트리로
\index{정리!정점정리}정점정리와
\index{정리!재고정리}재고정리를 넣고 싶으면...
```

어떤 단어를 index 엔트리로 지정하면 본문에는 나타나지 않으므로 그 단어를 중복해서 type해야 하는 것이 비효율적이라고 생각한다면 다음 예를 보라!

[3] 이 내용을 golatex.bat으로 만들어 사용하면 편리하다.

정리!점포정리

```
\newcommand\windex[2][\empty]{%
    \ifx#1\empty
        \index{#2}#2%
        \PrerenderUnicode{#2}%
    \else
        \index{#1!#2}#2%
        \PrerenderUnicode{#1#2}%
    \fi
}
\windex[정리]{점포정리}도 간단하게 인덱스의 작은 엔트리로 ...
```

위와 같이 하면 '점포정리'를 한 번만 type하여 인덱스의 작은 엔트리로 포함시킬 수 있다.[4]

\showindexmarktrue

인덱스에 포함된 단어를 해당페이지의 여백에 표시하면 인덱스 관리에 편리할 것이다. 이를 위해서는 (preamble에) \showindexmarktrue를 선언하면 된다. 최종 인쇄물에서 여백의 인덱스를 없애려면, 이를 comment 처리하면 된다. 간단하다.

[4] 새로운 명령어 \windex를 정의하는 이 코드는 김강수(2004)에서 가져 온 것이다.

5 캡션과 플로트

이 장은 (default 챕터스타일을 먼저 부른 후) hangnum 챕터스타일로 조판되었다. 여기에서는 `\hangsecnum` 을 선언하지 않았지만 이미 4.1.1에서 선언되었기 때문에 계속 장절번호가 왼쪽 마진까지 나온다.[1]

5.1 캡션

memoir에서는 캡션의 기능이 대폭 확장되었다. 먼저, 이 클래스에서의 기본값이 적용된 캡션의 모습을 보자. 그림이나 표의 내용 앞에 `\caption{...}`을 달면 그림이나 표 위에 캡션이 나오고, 아래에 달면 밑에 나온다.

figure here

그림 5.1: 밑에 캡션 달기

표 5.1: 위에 캡션 달기

table here

```
\begin{figure}[h]
  \centering \fbox{figure here} %내용
\caption[밑의 캡션]{밑에 캡션 달기}
\end{figure}

\begin{table}[h]
\caption{위에 캡션 달기}
  \centering \fbox{table here} %내용
\end{table}
```

[1] 어떻게 해야 `\hangsecnum`의 효과를 없앨 수 있을까?

5.1.1 캡션 스타일 살짝 바꾸기

캡션의 기본적인 형태는 [캡션이름] + [분리자] + [캡션 제목]으로 이루어져 있다. 이들의 모양를 원하는대로 바꿀 수 있다. 예를 들어, 다음과 같이 바꿀 수 있다. 이들을 플로트 환경 안에 코딩하면 그 플로트 환경에만 유효하고 밖에 하면 그 이후의 모든 플로트 환경에 영향을 미친다.

```
\captionnamefont{\Large\ssfamily} %defualt: {}
\captiondelim{-- } %default: {: }
\captiontitlefont{\itshape} %default: {} for normal
```

\precaption{...}과 \postcaption{...} 명령은 각각 캡션 전후에 처리된다. 이를 이용하여, 캡션 위에 선을 그으려면 \precaption{\rule{\linewidth}{0.8pt}\par}로 코딩하고 \postcaption{\rule{\linewidth}{0.4pt}}로 하면 캡션 아래에 선이 그어진다. 이 두 명령은 플로트 내에서 쓰이면 이 명령 다음에 오는 모든 캡션에 영향을 미친다. 그러나, 프리앰블이나 본문에 오면 그 효과는 파라미터 값을 바꾸기 전까지 계속된다.

결국, 캡션은 다음과 같은 형태로 식자된다.

```
\precaption
{\captionnamefont NAME NUMBER \captiondelim}
{\captionstyle\captiontitlefont THE TITLE}
\postcaption
```

\captionstyle[...]{...} 선언을 이용하여 캡션의 좌우정렬, 가운데 정렬 등을 조절할 수 있다. 아래의 예에서는 \captionstyle{\\}이 이용되었다. 이는 캡션이름과 캡션타이틀의 줄을 나누는 스타일이다.

표 5.2
REDESIGNED TABLE CAPTION STYLE

Some Table Here!

표 5.2의 코딩은 다음과 같다.

```
\begin{table}[htb]
\centering
\captionnamefont{\sffamily}
\captiondelim{}
\captionstyle{\\}
\captiontitlefont{\scshape}
\setlength{\belowcaptionskip}{10pt}
\caption{Redesigned table caption style} \label{tab:style}
  \fbox{Some Table Here!}
\end{table}
```


캡션 스타일을 preamble이나 .sty 패키지 파일에 적어 놓고 본문에는 간략하게 코딩하려면 다음과 같이 할 수 있다.

```
\makeatletter
\newcommand{\mycaption}[2][\@empty]{
  \captionnamefont{\sffamily\hfill}
  \captiondelim{\hfill}
  \captionstyle{\centerlastline\\}
  \captiontitlefont{\scshape}
  \setlength{\belowcaptionskip}{10pt}
  \ifx #1 \@empty \caption{#2}
  \else \caption[#1]{#2}}
\makeatother
```

먼저 위의 코딩을 preamble에 둔다 (\centerlastline의 효과에 대해서는 매뉴얼을 참고하라). 그리고, 다음을 코딩하면 표 5.3을 얻는다.

```
\begin{table}[htb]
\centering
\mycaption{Redesigned table caption style: again} \label{tab:style1}
\fbox{Some Table Here!: Again}
\end{table}
```

표 5.3
REDESIGNED TABLE CAPTION STYLE: AGAIN

Some Table Here!: Again

5.1.2 레전드

\legend{...} 명령은 플로트 내에서 \caption 명령과 무관하게 사용할 수 있다.

	C	D
C	$2,2$	$0,3$
D	$3,0$	$1,1$

그림 5.2– Prisoners' Dilemma

```
\begin{table}[htbp]
\captiondelim{-- } % 캡션제목 분리자 바꾸기
\legend{simple version}
\begin{center}
  \begin{tabular}{|c|c|c|}\hline
         & $C$    & $D$\\ \hline
  $C$ & $2,2$ & $0,3$ \\ \hline
  $D$ & $3,0$ & $1,1$ \\ \hline
  \end{tabular}
\legend{below}
\end{center}
\end{table}
\caption{Prisoners' Dilemma}
```

\caption과 마찬가지로 레전드 전후의 공백은 \abovecaptionskip과 \belowcaptionskip의 길이로 조절한다.

그 밖에 legend에 관한 기능이 많이 있다. (to be completed...)

5.1.3 서브캡션

subfigure 패키지

memoir 클래스에서 서브캡션을 구사하려면 subfigure 패키지를 불러서 함께 쓴다. 그러나 이 클래스에서는 기능을 확장하였다.

subfigure ONE
(a) subcaption 1

subfigure TWO
(b) subcaption 2

그림 5.3: subcaption test

그림 5.3에서는 \subcaption{...\label{...}}과 같이 함으로써 그림 5.3(a)와 (b) \subcaptionref를 얻었다.


```
\begin{figure}[htb]
\centering
  \begin{minipage}{0.3\textwidth}
    \fbox{subfigure ONE}
    \subcaption{subcaption 1\label{sf:sc1}}
  \end{minipage}
\hspace{3em}
  \begin{minipage}{0.3\textwidth}
    \fbox{subfigure TWO}
    \subcaption{subcaption 2\label{sf:sc2}}
  \end{minipage}
\caption{subcaption test}\label{fig:subcaption}
\end{figure}
그림~\ref{fig:subcaption}에서는 \verb+\subcaption{...\label{...}}+\와
같이 함으로써 그림~\ref{sf:sc1}\과 \subcaptionref{sf:sc2}\를 얻었다.
```

\subcation은 고정폭 환경에서 서브캡션을 넣는다. 한편, \subtop과 \subbottom은 내용의 폭을 이용하여 서브캡션을 식자한다. 두 그림의 서브캡션의 위치를 비교해보라!

그림 5.4는 세 개의 subfigure를 가지고 있는데, 바로 5.4(a)와 (b)와 (c)다.

\dotemph
\circemph
\subtop
\subbottom

SUBFIGURE ONE
(a) Subifgure 1

(b) Subifgure 2
SUBFIGURE TWO

SUBFIGURE THREE
(c) Subifgure 3

그림 5.4: A figure with some subfigures

```
그림 \ref{subfig:sf}는 세 개의 subfigure를 가지고 있는데,
바로 \ref{sf:1}\과 \subcaptionref{sf:2}\과 \subcaptionref{sf:3}\이다.
  \begin{figure}[htb]
  \centering
  \subbottom[Subifgure 1]{\fbox{SUBFIGURE ONE}\label{sf:1}}
  \hfill
  \subtop[Subifgure 2]{\fbox{SUBFIGURE TWO}\label{sf:2}}
  \hfill
  \subbottom[Subifgure 3]{\fbox{SUBFIGURE THREE}\label{sf:3}}
  \caption{Figure with some subfigures} \label{subfig:sf}
  \end{figure}
```

5.2 새로운 플로트 만들기

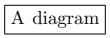
다음 예를 보자.

먼저 프리앰블에 다음과 같이 넣는다.

```
\newcommand{\diagramname}{Diagram}
\newcommand{\listdiagramname}{List of Diagrams}
\newlistof{listofdiagrams}{dgm}{\listdiagramname}
\newfloat{diagram}{dgm}{\diagramname}
\newfixedcaption{\fdiagcaption}{diagram}
\newlistentry{diagram}{dgm}{0}
```

그리고 본문에 다음의 내용을 식자한다.

Diagram 1: A diagram



You can see diagram 1 and diagram 2, which are just to show examples of using new float environment.

Diagram 2: Another diagram

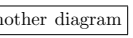


diagram 1과 diagram 2는 다음과 같이 코딩하였다.

```
\begin{diagram}[htb]
\caption{A diagram}  \label{diag1}
\centering  \fbox{Diagram 1}
\end{diagram}
You can see diagram~\ref{diag1} and diagram~\ref{diag2}, which are
just to show examples of using new float environment.

\begin{minipage}{.9\textwidth}
\fdiagcaption{Another diagram} \label{diag2} %%%
\centering  \fbox{Diagram 2}
\end{minipage}
```


원하는 곳에 `\listofdiagrams`를 두면, 그 곳에 'List of Diagrams'가 식자된다.

List of Diagrams

제 6 장

행과 열

이 장은 section 챕터스타일로 조판되었다. 여기에서도 \hangsecnum 을 선언하지 않았지만 이미 앞에서 선언되었기 때문에 장절번호가 여전히 왼쪽 마진까지 나온다.[1] 이 장에서는 array와 tabular환경 기능의 확장을 공부한다.

\verbfootnote

\footref

6.1 개요

```
\[ begin{array}[<pos>]{<preamble>} ... \end{array} \]
\begin{tabular}[<pos>]{<preamble>} ... \end{tabular}
\begin{tabular*}{<width>}[<pos>]{<preamble>} ... \end{tabular*}
\begin{tabularx}{<width>}[<pos>]{<preamble>} ... \end{tabularx}
```

⟨pos⟩ 옵션인자는 t(op), c(enter), b(ottom) 중의 하나이며, 기본 값은 c(enter)이다. 열의 개수와 형식은 ⟨preamble⟩에 의하여 지정된다. tabular*와 \tabularx는 ⟨width⟩ 인자로 폭의 길이를 제어할 수 있다.

6.1.1 Some Examples

성명	번호	비고
Abraham	305	월드컵 관전
Bob	200512	2번 마을 버스

[1] 어떻게 해야 \hangsecnum의 효과를 없앨 수 있을까는 여전히 의문이다. 각주[2]에서도 말했지만, 이 각주와 같이 memoir의 \verbfootnote{...} 명령을 사용하여 식자하면, 각주 내에서 \verb+\hangsecnum+를 바로 쓸 수 있으므로 편리하다! 여기서 각주번호 상호참조는 \footref{<label>}으로 식자했다.

```
\begin{center}
  \begin{tabular}{>{\large\bfseries}r|c >{\small\itshape}l} \toprule
  성명 & 번호 & 비고 \\ \midrule[0.2pt]
  Abraham & 305 & 월드컵 관전 \\
  Bob & 200512 & 2번 마을 버스 \\ \bottomrule
  \end{tabular}
\end{center}
```

b는 bottom에 정렬하도록 한다.	m은 baseline을 기준으로 middle에 ...	p는 (t가 아님에 주목) \parbox[t]{<width>}와 동일하다.

```
  \begin{center}
  \begin{tabular}{b{2cm}|m{2cm}|p{4cm}}\toprule
  b는 bottom에 정렬하도록 한다. &
  m은 baseline을 기준으로 middle에 ... &
  p는 (t가 아님에 주목)
  \verb+\parbox[t]{<width>}+와 동일하다. \\ \bottomrule
  \end{tabular}
  \end{center}
```

\tabularnewline

$0,1$	$-1,3$	$1,3$
$1,1$	$3,1$	$1,2$
$0,1$	$1,3$	$1,0$

여기에서 \\를 사용하지 않고(why? 작동하지 않아서) \tabularnewline을 사용하여 행을 바꾸었다. \\가 작동하지 않으면 \tabularnewline를 사용하자.

```
\begin{center}\marginpar{tabularnewline}
\begin{tabular}{|>{\centering$}m{3em}<{$}%
                |>{\centering$}m{3em}<{$}%
                |>{\centering$}m{3em}<{$}|} \hline
0,1 &-1,3 & 1,3 \tabularnewline \hline
1,1 &3,1  & 1,2 \tabularnewline \hline
0,1 &1,3  & 1,0 \tabularnewline \hline
\end{tabular}
\end{center}
```


새로운 컬럼 지시자를 정의해서 사용하면 편리하다. \newcolumntype

```
\newcolumntype{M}{>{\centering$}m{3em}<{$}}
```

컬럼 지시자를\newcolumntype{M}{>{$}>{\centering}m{3em}<{$}}와 같이 정의해도 마찬가지 결과를 얻을 수 있다.

$0,1$	$1,-3$	$1,3$
$1,1$	$3,1$	$-1,2$
$0,1$	$1,3$	$1,0$

```
  \begin{center}
  \begin{tabular}{|M|M|M|} \hline
  0,1 &1,-3 & 1,3  \tabularnewline \hline
  1,1 &3,1  & -1,2 \tabularnewline \hline
  0,1 &1,3  & 1,0  \tabularnewline \hline
  \end{tabular}
  \end{center}
```

dot나 comma를 기준으로 정리할 때는 컬럼지시자 D를 이용하는데, 다음과 같이 컬럼지시자를 재정의하여 쓰면 간편하다. 컬럼지시자 D

```
  \newcolumntype{d}[1]{D{,}{,}{#1}}

  \begin{center}
  \begin{tabular}{|d{-1}|d{-1}|d{-1}|} \hline
  0,1 &1,-3 & 1,3  \tabularnewline \hline
  1,1 &3,1  & -1,2 \tabularnewline \hline
  0,1 &1,3  & 1,0  \tabularnewline \hline
  \end{tabular}
  \end{center}
```

0,1	$1, -3$	1,3
1,1	3,1	$-1,2$
0,1	1,3	1,0

6.2 array 환경

표준 array 환경에서 확장된 기능은 `left`와 `right`의 괄호 쌍을 편리하게 나타낼 수 있다는 것이다. 여는 괄호와 닫는 괄호는 같은 모양이어야 한다. 한 쪽 괄호만 쓰고자 할 때는 period(.)와 함께 짝을 이루도록 하면 된다.

$$\left(\begin{array}{c|cc|c} & x_1 & x_2 & \\ & x_3 & x_4 & x_{12} \\ \hline & x_{21} & & x_{22} \\ & x_{31} & & x_{32} \end{array} \right)$$

```
\[
\begin{array}({cc})
  \begin{array}|{cc}|  x_1 & x_2 \\
                       x_3 & x_4    \end{array} & x_{12} \\
  x_{21} & x_{22} \\
  x_{31} & x_{32}
\end{array}
\]
```

주의할 것은 대괄호 `[` 와 `]`는 position 옵션과 함께 사용해야 한다. 그렇지 않으면 `[`가 여는 괄호인지 옵션의 시작인지를 구분할 수 없기 때문인 것으로 보인다.

$$\left[\begin{array}{cc} x_{11} & x_{12} \\ x_{21} & x_{22} \end{array} \right]$$

```
\[
\begin{array}[c][{cc}]
x_{11} & x_{12} \\ x_{21} & x_{22}
\end{array}
\]
```


6.2.1 표(Tables)

6.2.1.1 toprule, bottomrule, midrule, cmidrule

표에서는 수직괘선을 긋지 않는 것을 권장한다. 표준 LaTeX에서와는 달리, memoir에서는 수직괘선의 굵기 만큼 표의 가로길이가 늘어난다.

`\toprule`

`\midrule`

`\bottomrule`

`\cmidrule{n-m}`

표 6.1: Do not use vertical lines

Item		
Animal	Description	Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99

```
\begin{table}[htb]
\centering
\caption{Do not use vertical lines}
\label{tab:noverticallines}
  \begin{tabular}{@{}llr@{}}           \toprule
  \multicolumn{2}{c}{Item}  \\         \cmidrule(r){1-2}
  Animal & Description & Price (\$) \\ \midrule
  Gnat   & per gram & 13.65 \\
         & each     & 0.01  \\
  Gnu    & stuffed  & 92.50 \\
  Emu    & stuffed  & 33.33 \\
  Armadillo & frozen & 8.99 \\         \bottomrule
  \end{tabular}
\end{table}
```

```
\cmidrule[<wd>](<trim>){<n-m>}
```

`\cmidrule`은 n열부터 m열까지 midrule을 connect해준다. trimming을 위해서는 (r), (l), (lr)이 오는데, 예를 들면, (r)은 괘선의 오른쪽을 trimming 하라는 표시이다. 혹시라도 필요하다면 `[.8pt]`와 같이 괘선의 두께를 바꿀 수 있다. 괘선의 두께를 바꾸는 방법은 top, mid, bottomrule 경우에도 모두 같다.

그럴리 없겠지만, 혹시라도 두 개의 괘선이 필요하거든 다음과 같이 하면 된다.

```
\toprule \toprule
\cmidrule(lr){1-2}\morecmidrules\cmidrule{1-2}
```

행과 행사이에 2mm의 공백을 추가하고 싶으면, \\ 다음에 \addlinespace[2mm]를 넣으면 된다. 이 것은 2mm의 투명한 괘선을 긋는 것이라고 생각하면 된다.

\addlinespace

6.2.1.2 tabular 환경

tabular* 환경은 컬럼 사이의 공간을 변경함으로써 테이블 폭을 조절한다. tabularx 환경은 컬럼의 폭을 변경하여 테이블 폭을 변화시킨다.

다음에 공부하자.

제 7 장

newenvironment 연습

7.1 one example: environment

다음 소스는 새로운 환경을 정의하고, 정의된 새로운 환경을 사용하는 예이다.

```
\newcounter{EXAMP}[chapter]\newsavebox{\EXAMP}
\makeatletter
\newenvironment{emps}
{\stepcounter{EXAMP}%
\renewcommand{\theEXAMP}{\thechapter.\arabic{EXAMP}}%
\def\@currentlabel{\p@EXAMP\theEXAMP}%
\noindent \hrulefill\\ \textbf{예: \theEXAMP}}
{\noindent \hrulefill}
\makeatother

\begin{emps}\label{emps:first}
어쩌구 저쩌구 ...
\end{emps}
```

그 결과는 다음과 같다.

예: 7.1 어쩌구 저쩌구 ...

위의 정의에서 \p@EXAMP는 \pageref를 위한 정보이고 \theEXAMP는 \ref 명령을 위한 정보이다. LaTeX은 컴파일 도중 이 정보들을 foo.aux 파일에 기록한다.

그렇게 하면, (\ref{emps:first}에 의해) 7.1 또는 (\pageref{emps:first}에 의해) 41 페이지와 같이 번호를 참조할 수 있다.

7.2 another example: environment

```
\newenvironment{exer}{\noindent\hrulefill\\ \textbf{연습문제: }}%
        {\noindent \hrulefill}
\newenvironment{exerans}{\noindent\hrulefill\\ \textbf{연습문제답: }}%
        {\par\noindent \hrulefill}
```

위와 같이 '연습문제'와 '연습문제답'을 위한 환경을 정의하고 이를 다음과 같이 사용한다.

```
\begin{exer}
다음 문제를 풀어라.

. . .

\end{exer}
```

연습문제: 다음 문제를 풀어라.

1. $f(x) = e^x$의 $x = 0$에서의 선형근사를 구하라.

2. 차원이 같은 정사각행렬 A, B에 대하여, $AB = BA$가 사실인지의 여부를 밝혀라.

문제를 쳤으니, 이제 답을 치자. 물론 답을 위한 환경은 다음과 같이 이용한다.

```
\begin{exerans}
그런데, 답은 여기에 없다.
\end{exerans}
```

연습문제답: 그런데, 답은 여기에 없다.

參考文獻

\prebibhook: 이 참고문헌은 테스트를 위해서 작성한 것이므로, 실제 문서의 내용과 관련이 없다. 참고문헌 항목이 나열되기 전에 이 문장을 식자하기 위해 \prebibhook 을 이용하였다.

김강수 옮김, 『The LaTeX Memoir 매뉴얼』, *The Memoir Class for Configurable Typtesetting User Guide*, by Petter Wilson, 2004.

김강수, 『LaTeX-Memoir로 책만들기』, 2006.

Haeussler, Ernest F., Richard S. Paul, and Richard Wood, *Introductory Mathematical Analysis*, 11th ed., 2003.

Hughes-Hallett, Deborah, Andrew M. Gleason, and William G. McCallum, *Calculus: Single and Multivariable*, 4th ed., 2005.

Stewart, James, *Calculus*, 5th ed., 2005.

Sydsaeter, Knut, and Peter Hammond, *Essential Mathematics for Economic Analysis*, 2002.

\postbibhook: CTAN *is the Comprehensive TeX Archive Network and URLS for the several* CTAN *mirrors can be found at* http://tug.org. \postbibhook is used to place this sentence after all the bibitems. Note that \postbibhook command must be typed before \begin{thebibliography}.

찾아보기

4 LaTeX 문학적 프로그래밍: noweb

Editorial: 문학적 프로그래밍에 대해서는 작은나무 님과 smcho 님 두 분이 만드신 Literate Programming 페이지를 참고하세요. 4월에 smcho 님이 LaTeX으로 하는 문학적 프로그래밍 툴인 noweb을 소개하였습니다. 작은나무 님이 소개하신 CWEB은 지난 달에 이미 소개가 되었습니다.

조성민[5)]

4.1 noweb이란

noweb 은 LaTeX을 사용하는 cweb 이다. web/cweb 이 TeX을 대상으로 하는 것처럼 noweb 은 LaTeX 을 대상으로 한다. 즉, 기존의 모든 LaTeX의 패키지를 cweb 에서 제공하는 literate programming 과 함께 사용할 수 있다.

4.2 다운로드/인스톨

`http://www.eecs.harvard.edu/nr/noweb/`에서 소스를 다운로드 하여 컴파일 해주면 된다.

PC 에서 인스톨 하기 다음의 위치를 참고한다. `http://www.literateprogramming.com/noweb/nowebinstall.html`

맥에서 인스톨 하기 noweb 은 외부 프로그램을 불러서 추가 처리를 하는 기능이 있다. 외부 프로그램은 C 코드를 컴파일한 바이너리거나, awk 프로그램이거나 icon 프로그램이다. 그러므로 noweb 을 이용해서 다양한 기능을 이용하기 위해서는 반드시 awk 와 icon 이 깔려 있어야 한다.

- awk 는 기본적으로 깔려 있으나 tawk 라는 이름으로 호출되기 때문에 같은 이름의 alias 를 만들어 준다.
- icon 은 다음의 위치에 있다. `http://www.cs.arizona.edu/icon/`
  - 맥의 경우 바이너리가 있으므로 이를 인스톨 하여 사용하도록 한다.
  - icon 바이너리는 PATH상에 위치시켜준다.
- src 디렉토리에서 Makefile 을 다음과 같이 변경한다.

```
LIBSRC=icon
#LIBSRC=awk
# If you have no Icon compiler, but do have icont, make ICONC=icont
ICONC=icont
ICONT=icont
```

- 텍설정 디렉토리도 변경해 주어야 한다.

5) `http://faq.ktug.or.kr/faq/noweb`

```
TEXINPUTS=/Users/smcho/Library/texmf/tex/latex/
```

- 이후 `make; sudo make install;` 해주면 자동으로 깔린다. 반드시 모든 아이콘 소스코드가 깔리는 것을 확인해 주어야 한다!!!!

리눅스에서 인스톨 맥의 경우를 참고해서 인스톨 하면 될것으로 생각됩니다. – smcho

4.3 knoweb

기존의 noweb 을 개선한 스타일 파일, `http://www.k-online.com/~joer/noweb/knoweb.html` 에서 다운로드 받을 수 있다. 이 버전이 문제가 없고 더 안정적이므로 가능하면 knoweb.sty 를 사용하도록 한다.

4.4 예제 만들기

간단히 구구단을 해주는 예제를 만들어 보았다.

```
\documentclass{article}
\usepackage{noweb}
\usepackage{dhucs}
\begin{document}

\section{구구단 만들기}
구구단을 해주는 프로그램을 작성해 보자.

@
가장 높은 레벨에서는 다음과 같이 처리할 수 있다.
<<nine.c>>=
<<headers>>
<<engine>>
<<main>>
<<print>>
<<test>>
@

상태는 [[status]]로 표시가 가능하다.
<<headers>>=
/* 헤더파일 */
#include <stdio.h>
#include <assert.h>
@

<<engine>>=
/*
   DoxyGen string
*/
int engine(int x, int y)
```

```
{
  assert(x >= 0 && y >= 0);
  return (x * y);
}
@

<<print>>=
/*
  Write down the result.
*/
int print(int x, int y, int z)
{
  printf("(%d) X (%d) = (%d)\n",x,y,z);
}
@

<<main>>=
/*
  Main Routine
*/
int main()
{
  int i, j;
  for (i = 1; i < 3; i++) {
    for (j = 1; j < 3; j++) {
      int z = engine(i,j);
      print(i,j,z);
    }
  }
}
@

<<test>>=
/*
  Test routine
*/
int test()
{
  printf("testme\n");
}
@
\end{document}
```

4.5 사용법

텍파일 얻기

```
noweave -delay nine.nw > nine.tex
```

소스코드 얻기

```
notangle nine.nw -Rnine.c > nine.c
```

4.6 결과

텍소스를 컴파일 하면 pdf 를 얻을 수 있다.
소스코드를 컴파일 하면 프로그램을 얻을 수 있다.

```
$ gcc nine.c -o nine
$ ./nine
(1) X (1) = (1)
(1) X (2) = (2)
(2) X (1) = (2)
(2) X (2) = (4)
```

4.7 코드 청크

noweb 은 <<>>로 상징되는 코드 청크들을 하나로 모아주는 기능을 제공한다. `\nowebchunks` 명령을 이용한다.

4.8 인덱스

noweb 은 어떠한 정의가 어디에서 사용되었는지를 알려주는 index 를 만들 수 있다.

인덱스 정의

`[[main]]` 과 같이 사용하면 main 이 인덱스로 기록된다.

인덱스 지정

`@ %def main` 을 코드청크 다음에 기록하면 인덱스와 코드청크가 연결이 된다. main 은 인덱스로 사용할, 미리 정의된 값이다.

인덱스 생성

`\nowebindex`를 사용하여 생성한다.
noweb 명령에서 -index 를 추가한다.

4.9 책갈피

보통의 hyperref 를 사용하는 것 만으로도 코드 청크 끼리의 연결이 자동으로 생성된다.

```
\usepackage[pdftex,colorlinks,backref,bookmarks
  ,pdftitle={knoweb.sty}
  ,pdfauthor={smcho}
  ,pdfpagemode={UseOutline}
  ,bookmarksopen={true}
  ]{hyperref}
```

4.10 큰 프로그램 만들기

웹은 하나의 nw 파일에서 여러개의 소스파일을 만들어 낼 수가 있다. -R 옵션을 사용하면 된다.

또한 웹은 여러개의 nw 파일을 만들고, 각각을 tex 으로 변환한 다음에 `\input` 을 이용해서 하나의 '큰' 텍 파일을 만들어 낼 수 있다.

이 두가지를 종합하면 아주큰 프로젝트를 웹으로 만들어 낼 수가 있다. 그러나 이 경우 make 파일도 동시에 만들어야 한다.

다음에 Makefile 의 예를 들었다. make tex 을 이용해서 현재 디렉토리에 있는 모든 nw 파일에서 텍 소스와 매트랩(.m) 을 자동으로 생성해 내고, 이후에 make명령을 이용해서 pdf 를 만드는 구조로 되어 있다.

기존의 Makefile 에 비해서 셸의 for 기능을 이용해야 한다. 이것은 오브젝트 파일을 모아서 하나의 실행파일을 만드는 보통의 C 프로그래밍 과는 다르게, 하나의 tex 파일에 다수의 .tex 파일을 include 하는 구조를 이용해야 하기 때문이다.

만일, Make 고유의 기능을 이용해서 변경된 파일만을 처리하기 바란다면 일일이 abc.m : abc.nw 처럼 소스파일의 의존성을 기록해 주면 된다. 예에서는 자동으로 처리되는 방법을 사용했다.

```
SHELL=/bin/sh
NOWEAVE=noweave
NOTANGLE=notangle
NODEFS=nodefs
PDFLATEX=pdflatex

TARGET=codeanalysis
NWS = $(wildcard *.nw)
NAMES = $(NWS:.nw=)
TEXS = $(NWS:.nw=.tex)

.SUFFIXES: .nw .tex .dvi .html .pdf .m
.SECONDARY: $(TEXS) # Don't delete the tex file
.PHONY: tex

all: $(TARGET).pdf

test:
@echo $(TEXS)

pdf:
$(PDFLATEX) $(TARGET).tex

tex:
@for t in $(NAMES); do \
echo "processing $$t.nw"; \
$(NODEFS) -autodefs c $$t.nw > $$t.defs; \
$(NOWEAVE) -delay -filter btdefn -autodefs c -indexfrom $$t.defs $$t.nw > $$t.tex; \
$(NOTANGLE) -R$$t $$t.nw > $$t.m; \
done
```

```
$(TARGET).pdf : $(TARGET).tex
$(PDFLATEX) $< -o $@

.nw.tex:
$(NODEFS) -autodefs c $*.nw > $*.defs
$(NOWEAVE) -delay -filter btdefn -autodefs c -indexfrom $*.defs $*.nw > $@


sclean:
rm -f *~  *.dvi *.log *.html *.pdf *.m *.out *.defs

clean:
rm -f *~  *.dvi *.log *.html *.pdf *.m *.out *.defs *.tex *.aux
```

4.11 기타

- notangle 이 만들어 주는 소스코드는 조금 답답한 느낌을 준다. 이경우 bcpp 등을 이용하여 소스를 예쁘게 변환할 수 있다. `http://dickey.his.com/bcpp/bcpp.html`
- `-ylcnc` 를 사용하면 커멘트가 코드와 잘 연결된다.

4.12 예제

유명한 wc 를 noweb 을 이용해서 만들어 본다.

다운로드 wc.zip

컴파일

```
nodefs -autodefs c wc.nw > wc.defs
noweave -filter btdefn -autodefs c -indexfrom wc.defs wc.nw > wc.tex
pdflatex wc.tex
```

으로 pdf 를 생성할 수 있다.

```
notangle wc.nw > wc.c
```

로 C 소스코드를 생성할 수 있다.

결과 wc.pdf 예제에서, 인덱스 및 코드 정크가 pdf 상에서 완벽하게 연결되어 지는 것을 볼 수 있다. 한글도 dhucs 를 사용하는 경우에는 문제없이 잘 된다.

5 yhchoe의 입문자 코너 : WinEdt Tree 편집

Editorial: yhchoe 님의 WinEdt Tip 프로젝트는 LaTeX 입문자에게 많은 도움을 주는 코너입니다. 4월 중에 갱신된 페이지 하나를 소개합니다. 비록 WinEdt이 hangul-ucs를 사용할 수 없다는 점 때문에 홀대받고 있는 느낌이 있지만 멋진 TeX 사용환경임에 틀림없습니다.

최영한[6]

5.1 WinEdt Project Tree

WinEdt에는 Project Tree라 하여 책이나 긴 글을 쓸 때 편리한 기능[7]이 있습니다. Project Tree란 그림 1과 같이 WinEdt 창을 나누어 왼쪽에 작업 문서의 구조를 수형도(樹型圖: Tree)로 나타나게 하는 기능입니다.

그림 설명 lshort-kr에서 lshort-kr TeX 원본 파일을 클릭하면 "LaTeX 2ε 입문"의 소스 파일인 `lshort-kr.tex`이 들어 있는 `lshort-kr-src.tar.gz`이 다운로드됩니다. 이것을 풀면 lshort-kr 폴더가 생기고 그 속에 있는 많은 파일들이 있습니다. WinEdt가 깔려 있는 경우 이들 중에 `lshort-kr.tex`을 클릭하면 WinEdt 창에 뜹니다. 이때 WinEdt의 단추들 중에 녹십자가 있는 단추에 커서를 가져가면 "Set Main File"이라는 풍선도움말이 뜹니다. 이 단추를 누르면 WinEdt는 WinEdt 창에 활성화되어 있는 파일을 읽고 Project Tree를 만듭니다. TeXify한 다음 "Project Tree" 단추를 클릭하면 그림의 왼쪽에 나타난 것과 같이 Project Tree가 나타납니다. 그림에서처럼 Project Tree에는 파일들(title, contrib, overview, things, typeset, math, lssym, spec, custom, biblio)이 나타나는데 이 파일들은 메인 파일(위의 그림에서 오른 쪽에 열려 있는 파일)에서 `\include{title}`, `\include{contrib}` 등으로 편집되어 있기 때문입니다. 또 맨마지막에 있는 `aterword.tex`은 "input aterword.tex"으로 불렀기 때문입니다. 컴파일 전과 컴파일 후의 Tree에 나타나는 것이 다릅니다. 컴파일한 다음 Project Tree를 열면 TOC 파일 내의 Tree도 보여 줍니다(그림 참조).

5.2 include와 input의 차이

Q: (From KTUGOperate:15073) 안녕하세요? 다음과 같은 것이 궁금하네요. 예를들어 *example.tex*라는 파일이 있구요. 이것을 *main*파일에서 *includeexample*해서 부르는 거랑 *inputexample.tex*해서 호출하는 것과 어떤 차이가 있는지요. *include*를 쓰면 따로 따로 컴파일을 하는것 같아서 해당 디렉토리에 *example.aux*라는 파일도 생기던데요. 이렇게되면 디렉토리 내부가 지저분해져서, 저는 *input*을 주로 사용하는데요. 위의 질문과 각각의 장단점을 아시면 부탁 드립니다. 그럼

A1: lshort-kr.pdf의 15쪽에 보면 "`\include{filename}`으로 삽입되는 부분을 처리할 때는 새로운 쪽에서 시작하고 `\input{filename}`으로 삽입되는 부분을 처리할 때는 페이지 조절이나 문자 추가를 전혀 하지 않고 삽입된 위치에서 한다"고 하였습니다. 그래서 `\include{}`는 여러

6) `http://faq.ktug.or.kr/faq/WinEdtTip`

7) `http://www.ktug.or.kr/jsboard/read.php?table=contrib&no=46`

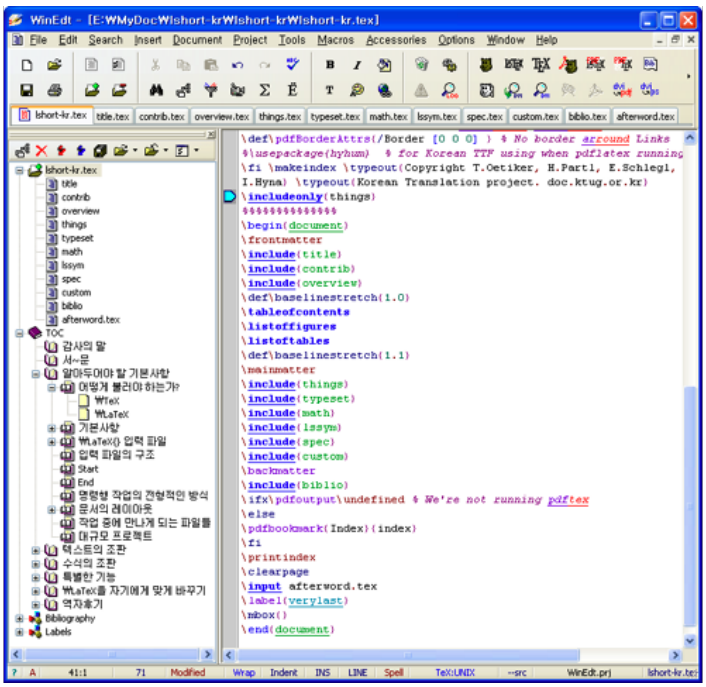


그림 1: WinEdt의 Project Tree

개 썼더라도 `\includeonly{filename}`[8]에 한 개 또는 몇 개만 선택하여 넣으면 출력 파일에서는 `\includeonly{}`에 포함한 것만 보여줍니다. 쪽 번호, 장-절 번호, 표 번호, 그림 번호 등 여러 가지 번호(자동 번호 매기기에서 주어지는 번호)는 바뀌지 않습니다. "쪽 레이아웃"(그림의 위치 표의 위치 등)도 바뀌지 않습니다. 각각의 include 파일은 마치 독립적인 파일처럼 컴파일됩니다. 그래서 aux 파일도 별도로 생깁니다. 만약 `\input` 파일은 컴파일 때 제외하고 싶으면 일일이 주석(%) 처리하여야 합니다. 이 때 "쪽 레이아웃"(그림의 위치, 표의 위치 등)도 바뀝니다. input 파일은 마치 메인 파일의 일부분처럼 컴파일됩니다. 그래서 별도의 aux 파일도 생기지 않습니다. 또 `\input`을 쓸 때는 일일이 확장자를 쓰는 것으로 보아서 확장자가 tex이 아니라도 될 것 같습니다.

A2: 참고문헌만들기에서 chapterbib 패키지를 쓰서 "장(章)별 문헌 목록" 만들기를 할려면 `\include{}`를 쓰라고 하였습니다.

5.3 부분 컴파일 기능

“부분 컴파일 기능”은 “Project Tree”의 기능은 아니지만 긴 소스 파일을 입력할 때 쓸 수 있는 편리한 기능입니다. 현재 작업(입력 또는 교정)하고 있는 부분이 잘 되었는지 부분적으로 컴파일하는 기능(Compile Selected Block (Shift+Ctrl+C): 컴파일하고 싶은 부분을 선택(파란 바탕에 흰

8) 명령어 `\includeonly{filename}`는 프리앰블에서만 쓸 수 있습니다. 만약 `\include{}` 앞에 %를 넣어 해당 파일을 주석하면 그 파일은 Project에 포함되지 않기 때문에 쪽 번호, 장-절 번호, 표 번호, 그림 번호 등 자동 번호 매기기에서 주어지는 번호는 새로운 상황에 따라 바뀝니다.

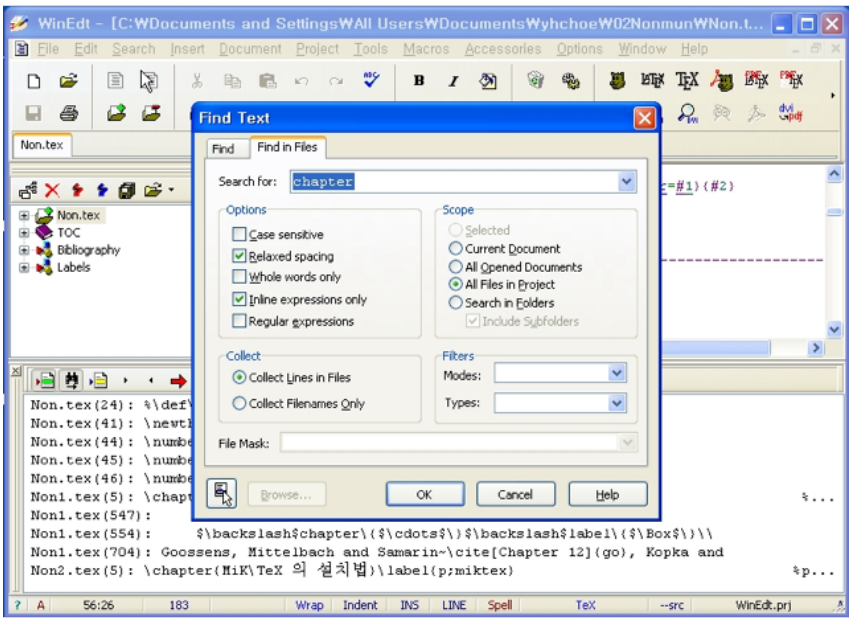


그림 2: 여러 파일에서 찾기

글씨로 변합니다.)하고 황금 사자머리(TeXify) 단추 바로 밑에 "흰 사자머리" 단추를 누르면 그 부분만 컴파일됩니다. 이때 자동 번호 매기기(그림 번호, 수식 번호 등)는 제대로 되지 않습니다. 입력이 제대로 되었는지만 체크합니다. 전체 적인 Project에는 변함이 없습니다.

5.4 Project Tree와 관련한 기능

includeonly

문서를 만든 도중에 매번 전체 문서를 컴파일하는 것은 매우 귀찮은 일입니다. 이 때는 위 그림의 오른쪽 윗줄에 있는 것 처럼 메인 파일에 `\includeonly{}`을 입력하여 두고 {} 속에 한 개 또는 몇 개의 파일만 넣고 컴파일할 수 있습니다. 하나도 넣지 않으면 메인 파일만 컴파일됩니다. 만약 전체를 컴파일하고 싶으면 `\includeonly`를 주석 처리(%를 앞에 붙이는 일)하면 됩니다.

가령 메인 파일에 장-절의 명령, 머리말-꼬리말, 목차 만들기, 인덱스 만들기, 어떤 환경의 장치 등을 바꿔 다시 컴파일하면 각 include 파일에 들어 있는 aux 파일이 다시 만들어지지 않기 때문에 장치의 변경에 따른 레이아웃이 바뀌지 않습니다. 그래서 이 경우 include 파일에 딸린 aux 파일을 지우고 전체적으로 다시 컴파일하여야 합니다.

Find in Files

Q: (From KTUGOperate:17676) 어떤 *string*을 검색할 때 모든 연결문서들을 다 열어두지 않은 상태에서 *include*나 *input*으로 연결된 문서까지 함께 검색하도록 할 수 있나요? *Find* 옵션에 *"All files in project"*라는 게 보이긴 하는데 *project*에 어떤 파일들이 속하는지 미리 그 리스트를 수작업으로 등록시켜주는 작업이 먼저 필요할 것같은 생각도 드는데...

A1: 리스트를 수작업으로 등록시켜주는 작업을 할 수 있는지 모르겠습니다. 한 번만 “Main File”을 지정(Set Main File)하고 아래 그림과 같이 Project Tree를 열어 두고 Find in Files 기능을 쓰면 됩니다. 그 다음부터는 Project Tree를 열어 두지 않아도 됩니다. 그림 2를 참고하세요.

5.5 그 외

WinEdt에는 컴파일할 때 어떤 특별한 작용을 하지 않지만 Project Tree에 수형도(樹型圖: Tree)의 가지(branch)로 나타나게 하는 기능이 있습니다. 예를 들어 설명하겠습니다. 메뉴에서

Document -> Current Work (Samples) -> PhD Thesis

를 클릭하면 WinEdt 창에 Thesis.tex 파일이 열립니다. 이 파일의 거의 끝부분에 가면

```
%GATHER{xBib.bib}   % For Gather Purpose Only
%GATHER{Thesis.bbl} % For Gather Purpose Only
```

이 있습니다. 이것이 Project Tree에 xBib.bib와 Thesis.bbl이 나타나게 하는 장치입니다.
`Thesis.tex` 파일은 Project의 한 예입니다.

6 공주대학교 TEX 포럼 후기

이호재[9)]

공주대학교 인문사회 포럼

- 한글 LaTeX의 활용 가능성
  1. 조진환, TEX과 디지털 타이포그래피
  2. 남상호, 경제 분석과 LATEX 활용
  3. 김강수, LATEX 한글 구현 성과와 전망
- 공주대학교 경제통상학부 주관 (학부장 조인성 교수)
- 2006 년 4 월 26 일
- 공주대학교 인문사회관 111 호 세미나실

6.1 조진환, TEX과 디지털 타이포그래피

구텐베르크가 15 세기에 개발한 인쇄 시스템(금속 활자, 잉크, 프레스)은 상업적 대량 생산을 가능케 하였다. 이 시스템은 18 세기까지 Lynotype이 개발될 때까지 큰 변화 없이 지속되었다. Lynotype은 준비된 활자들을 자판기를 이용하여 줄 단위로 배치하는 시스템이었다. 이 시스템은 구텐베르크의 시스템에 비해 최소 4 배의 속도 향상을 가져 왔다. 20 세기에 들어 컴퓨터의 발명으로 가능해진 Desktop Publishing은 이전 세기와는 비교할 수 없을 정도의 생산성의 혁명을 일으켰다. 여기에 우리는 그것을 크게 3 가지로 나눌 수 있다.

1. 워드 프로세서: MS Word, 아래아 한글
2. 레이아웃 프로그램: 익스프레스, 인디자인
3. TEX

워드 프로세서는 사용하기 손쉬운 프로그램이지만 안정성에 문제가 많이 있고 아름다운 문서를 만들기에 한계가 있다. 데이터베이스와의 연동이나 자동화를 할 수 없다는 것도 단점으로 지적할 수 있다.

레이아웃 프로그램은 배우는 데에 아주 오랜 시간이 걸리고 고가이다. 매우 아름다운 문서를 만들 수 있지만 막대한 노동 집약에 의해서만 가능하다. 워드 프로세서와 마찬가지로 데이터베이스와의 연동이나 자동화는 불가능하다.

텍의 조판 능력을 능가하는 것은 없다. 공개 소프트웨어이다. 버그프리이다. 데이터베이스와의 연동과 자동화가 가능하다. (뒷풀이에서 김병룡님이 엑셀 파일로부터 데이터를 추출하고 스타일을 입혀 만든 문서를 보여주심)

9) http://faq.ktug.or.kr/faq/%B8%F0%C0%D3

호재 주석: 조진환님은 맥 노트북을 이용하여 발표하였다. 키노트라는 프로그램으로 만든 그의 발표문에서 최초의 페이지 넘김이 cube transition으로 일어나자 몇몇 학생들이 작은 탄성을 질렀다. 뒤에 김강수님의 발표문에서도 멋진 transition이 일어나자 학생들은 비슷한 반응을 보였는데 나는 이 대목에서 일반 사용자들을 위해 몇 가지 충고를 드리겠다. 조진환님은 순전히 텍만을 이용해도 아주 환상적인 Presentation을 만들 수 있다며 다른 발표문들을 예로서 보여주셨다. 이 말은 사실이면서도 거짓이다. 왜냐하면 우리가 텍이라 일컬을 때 3가지 의미로 말한다. 하나는 조판 언어, 하나는 컴파일러, 마지막 하나는 텍 배포판이다. 일반 사용자들은 조판 언어나 컴파일러로 받아들이겠지만 여기에서 조진환님은 텍 배포판의 의미로 말한 것이다. 실제 예로 보여주신 것들은 PDF Special과 Metapost 처리를 위해 ConTeXt로 만든 것이다. 이는 ConTeXt 뿐만 아니라 PDF Special과 Metapost도 알아야 함을 의미한다. 김강수님은 AcroTeX을 이용하여 Layer 효과가 들어가게 만들었다. 이 분들이 '된다' 혹은 '쉽다'라고 말씀할 때 최소한 열 배의 부담을 더하여 듣기 바란다. 실제 그 환상적인 발표문들을 보게 된다면 그 아름다움에 매료되어 도전하고픈 충동에 휩싸이기 십상이지만 Beamer에 만족함이 현명한 선택임을 굳이 권하고 싶다.

6.2 남상호, 경제 분석과 LaTeX 활용

1. 첫 직장인 KDI에서 문서와 관련하여 많은 고생을 하였다. 전두환 정권 시절이었는데 한 친구는 全두환을 金두환으로 표기하여 안기부에 끌려가서 치도곤을 당하였다.
2. 미국에 유학 가서 텍을 알게 되었고 그것을 이용하여 많은 문서를 작성하였다.
3. 귀국 후에 텍을 이용하려 했지만 한글 사용에 어려움이 많았다.
4. 조진환과 김강수로부터 많은 도움을 받았다.
5. 박영사와 출판 계약한 **현대경제변동론**의 조판이 마음에 들지 않아 내가 직접 텍을 이용하여 조판하였다.

호재 주석: 출판사의 편집자들이 밥줄 끊는 노릇이라며 뒤에서 분명 욕했을 것이다.

6.3 김강수, LaTeX 한글 구현 성과와 전망

1. 워드프로세서도 스타일만 일관되게 하면 쓸만한 문서를 만들 수 있다.
2. Visual markup과 plain text
3. UHC와 UCS
4. 다양한 조판 사례

호재 주석: 학생들이 전혀 이해하지 못하는 눈치였다. 마치 '아햏햏가 도대체 뭐가 문제지?'

6.4 뒷풀이

- 토지 (모임을 늘 여기에서 하고 싶을 정도로 아주 맛있는 한정식, 여기서 조인성 교수님께 감사를 표하지 않을 수 없다.)
- 카페 (금강 옆에 있는 아주 멋지고 다소 비싼)

조진환: hLaTeXp의 개발에는 내로라 하는 인력들이 참여하였고 그래서 더 없이 잘 만들어진 시스템이었지만 폰트 라이센스로 인해 더 이상 사용할 수 없다. hLaTeX과 비교하면 NFSS

가 없다는 것이다. hLaTeX은 이에 비해 더 좋은 알고리듬이라고 할 수 없지만 UHC 글꼴을 제공한 것이 기막힌 발상이었고 최대 장점이었다.

김강수: 정부에서 금속활자로 200부 정도 찍어 감영에 내려보내면 감영은 그것을 풀어서 목판본을 만든다...(조판에 대한 애착과 지식의 깊이를 가늠조차 할 수 없게 하는, 그래서 오히려 할아버지들의 옛이야기를 듣듯이 편안한 마음으로 들을 수 있는)

조진환: 파이썬이든 텍이든 미적분학이든 수행할 프로젝트가 없으면 학습이 제대로 이루어지지 않는다.

그밖에 수학, 음악, 박물관, 맥킨토시(언제나 빠지지 않는), Spacing, 한자 서체, 한글 처리를 위한 Primitive, 일러스트레이터, xypic 등등이 이야기되었다.

KTUG 모임에는, 최영한 교수님, 늦게오신 현범석 님, 조인성 교수님, 멀리서 오신 김병룡 님, 역시 멀리서 오신 이호재 님, ChoF, Karnes 가 참석함. ChoF님의 한마디: "서울 모임보다 대전 모임이 분위기가 더 좋네" 아무래도 비싼 것을 드신 때문이 아닐까...^^

김병룡 님은 악보와 주소록 조판한 것을 보여주셨고, 이호재 님은 일러스트레이터가 싫증났다는 폭탄선언과 위성 공부에 대한 얘기...현범석 님은 "20일 걸려서 돌린 프로그램"에 대한 얘기와...emacs에서 유니코드 쓰기 힘들다는 불평...emacs를 최신 버전으로 바꾸세요. 조인성 교수님이 나도 한번 매킨토시를 써볼까.. 하시자 ChoF 님이 유닉스/리눅스에 익숙한 분들은 맥이 정말 좋은데...아니라면 한번 더 생각해 보심이...라고 하심.

Colophon

KTUG : Korean TEX Users Group.

http://www.ktug.or.kr

KTUG Faq : http://faq.ktug.or.kr/faq/

편집 및 PDF 제작 : Karnes

2006년 5월 1일

• 이 책자는 Memoir, Hangul-ucs, memhangul-ucs로 조판되었습니다. MinionPro와 아시아 신명조를 본문 글꼴로 하여 제작되었으며, pdf 파일의 임베딩은 pdfpages 패키지를 이용하였습니다.