Reading *TEXbook* (5) : 줄바꿈 알고리듬

작은나무

2006년 5월

차 례

5월의 학습 목표

작은나무 2006-04-30T23:35:02

5월 부터는 그 달의 학습 목표를 세우고 체계적으로 차례차례 읽어나가기로 했습니다.

그동안 관심있는 것만 읽고 공부해왔는데, 그러다보니 특정분야에 대해서는 아직 하나도 모르겠고, 이는 원래 목표인 텍소스(텍: 더 프로그램)를 읽는데, 걸림돌이 되고있습니다.

텍은 다음과 같은 뛰어난 능력을 가지고 있다고 합니다.

- 줄 바꿈(Line breaking) 알고리듬 (14장)
- 쪽 나눔(Page breaking) 알고리듬 (15장)
- 하이프네이션 알고리듬 (부록. H)
- 수식 조판 기능 (16 19장)

5월 달은 첫번째 "줄 바꿈 알고리듬"(Line breaking algorithm)을 목표로 하고, 매페이지마다 한줄한줄 꼼꼼히 읽어 나가겠습니다. 어려운 부분은 다른 참고서도 이용하겠습니다. (참고서라봐야 작은나무가 가지고 있는 책은 *TeX by Topic*뿐입니다. 그것도 pdf로만. 어쩌면 그책이 더 어려울 수도 있습니다. ^^;) 그리고 주어진 연습문제를 모두 해결하는 것은 기본입니다. 읽어나가다가 막히면, 해결될 때까지 앞으로 나가지 않겠습니다. 다시 말하면, 이 "텍북 읽기"가 중간에서 끝날 수도 있다는 말입니다. 그런 일이 일어나지 않도록, 작은나무가 엉터리로 이해하여 틀린 곳으로 가고 있으면, 길을 바로 잡아주시고, 힌트나 영감도 많이 불어 넣어 주시길 바랍니다. (거의 협박 수준입니다. -_-;)

감사의 말 이 글을 쓰기 쉽도록 언제나 페이지를 만들어 주시고, 환경을 제공해주시는 [Karnes] 님께 진심으로 감사드립니다. [Karnes]님을 봐서라도 중간에 그만두는 일은 없어야 겠습니다. 그만 둘 수도 있다는 말은 농담이었습니다. :)

▶ 기대가 큽니다. *PlainTeX*에 대한 주의를 환기해주신 것, 프로그래밍과 *TeX*에 대한 독보적인 영역을 개척하고 계신 것에 감사드립니다. 작은나무 님 덕분에 *KTUG Faq*가 풍성해지고 있습니다. 제가 하는 일이 원래 게시판 관리를 지우는 거 하고 *Faq*에 글 올리시는 분들 독촉하는 거지요. 그런 일을 하는 사람을 "디렉터"라고 부른답니다. – *[Karnes] [[DateTime(2006-05-01T02:55:25)]]*

▶ *"'O Director! my director!"' ;)* 출처: *O Director! my director!*[1)]

---

1) http://www.blindwave.com/dead_poet_society/board/view.php?blindwave=ap2\&id=6\&page=1\&spg=1

1 줄 바꿈 알고리듬 익혀야 하는 이유

작은나무 2006-05-01T01:53:41

91쪽: 입력으로 주어진 일련의 기 다란 문자열들을 보기 좋은 줄 바꿈을 통하여 여러 줄들로 나누는 것은 조판 시스템이라면 반드시 갖추어야 하는 기능이다. 텍은 이 모든 것을 자동으로 해주고 그 방법에는 매우 흥미로운 알고리듬이 있다.

줄 바꿈에는 badness, overfull, underfull, \tolerance 같은 것들에 대한 지식이 필요한데, 이는 다음을 참고한다.

- Penalty[2)]
- Badness와 tolerance[3)]
- Demerit[4)]

위에 대한 내용들은 다시 언급할 기회가 있을 것이다. 사실, 위의 내용들은 우리가 지금 공부하고 있는 14장에서 발췌해서 작성한 글 이기도 하다.

텍은 주어진 문단에 대해서 badness를 최소화하여 줄바꿈을 하는 엄청나게 좋은 방법 (absolutely best way)을 가지고 있고, 그 방법은 물론 텍(컴퓨터)이 하므로 우리에게는 자동적인 방법이 되겠다. 텍이 자동적으로 알아서 그것도 매우 잘 줄 바꿈을 해준다는데, 우리들이 알 필요가 있을까? 알 필요가 있다. :) 컴퓨터가 아무리 좋은 방법으로 문제를 해결해 준다고 하더라도 우리들, 변덕장이 혹은 개성이 저마다 다른 사람들에게는 맘에 들지 않을 수가 있다. 혹은 글의 구성 혹은 전개상 전혀 쌩뚱 맞은 곳에서 줄을 바꿔야 하는 경우도 있다. 그러한 때에는 줄 바꿈을 텍에게 맡기는 것이 아니라, 수동적으로 우리들 자신이 해야한다. 그 경우 우리들이 약간만 텍을 도와주면, 텍은 우리들 맘에 꼭드는 줄 바꿈으로 우리에게 보답한다. 14장은 텍의 자동화 방법을 익히는 것도 되지만, 우리가 텍을 도울 수 있는 방법을 익히는 것도 된다.

차례에 의해서 다음 글에서 공부할 내용은 ~ 표기되는 tie 이다. PlainTeX에서 이 타이의 기능은 매우 유용해서 이 것을 언제 어느 곳에 적절히 넣을 수만 있다면, 기본적인 TeXnical 타이피스트는 졸업을 한 셈이고, 곧 고급자 코스(the select group of Distinguished TeXnician)로 들어갈 수있다는 것을 의미할 정도라고 한다.

2 틸데 사용법(1)

작은나무 2006-05-01T09:41:25

91 92쪽: 우리가 입력 파일에 ~를 쓴다는 의미는 보기에는 스페이스바를 눌러서 간격(공백)을 하나 만드는 것과 다를바가 없다. 그런데 그러한 간격과 큰 차이점은 텍은 ~에서는 줄 바꿈을 하지 않는다는 것이다. 상식적으로 줄 바꿈은 간격과 같은 글루에서 이루어진다. 주의할 것은 ~ 다음에 간격을 넣지 말아야 한다는 것이다. 만약 ~ 다음에 간격을 넣으면, 그 간격도 의미가 있어서 두 개의 간격을 넣은 효과를 내기 때문이다. 즉, 우리가 예상하는 간격보다 더 큰 간격이 생긴다.

---

2) http://faq.ktug.or.kr/faq/LittleTree/ReadingTeXbook/2006-03#18

3) http://faq.ktug.or.kr/faq/LittleTree/ReadingTeXbook/2006-03#90cecc9f62df8b348dc17a6cbb3df42f

4) http://faq.ktug.or.kr/faq/LittleTree/ReadingTeXbook/2006-03#74e0be85ae546e892989b8d590ad52a4

마찬가지로 입력 라인 맨 끝에 ~를 넣으면, 보다 넓은 간격이 생기는데, 이는 ~ 다음의 <리턴> 이 말을 하기 때문이다.

타이가 쓰이는 곳은 어떤 곳일까? 문장의 끝을 나타내지 않는 마침표 뒤에 ~를 붙이면 좋다. 즉 생략, 축약을 나타내는 마침표 뒤에 말이다. 다음을 보자

```
Mr. Drofnats was happiest when he was at work typesetting documents.

Mr.~Drofnats was happiest when he was at work typesetting documents.
```

Mr. Drofnats was happiest when he was at work typesetting documents.

Mr. Drofnats was happiest when he was at work typesetting documents.

첫번째 Mr. 다음에 약간의 간격이 더 들어간 것을 눈으로 확인 할 수 있다. 이것은 문장의 마침표 뒤에 약간의 간격을 더 넣기 때문이다. 하지만 Mr. 는 문장의 끝이아니라 축약 표현이므로 Mr.의 . 다음에 약간의 간격이 오는 것은 적절하지 않다. 이 경우에 Mr.~와 같이 ~를 사용하는 것이다. 이와 관련하여 알아 둘 플레인텍의 매크로가 있다. \frenchspacing과 \nonfrenchspacing. \frenchspacing 이라고 하면 마침표 뒤에 약간의 간격을 넣지 않는다. 이는 특별한 경우에 사용되고, 플레인텍의 기본은 \nonfrenchspacing 이다.

다시 돌아와서 타이는 어디에 사용하면 좋을까?

- 아래와 같이 문서의 이름을 가진 일부를 가리킬 때,

  ```
  Chapter~12          Theorem~1.2
  Appendix~A          Table~\hbox{B-8}
  Figure~3            Lemmas 5 and~6
  ```

  Lemmas 뒤에는 ~ 가 없다. 왜냐하면 Lemmans 다음에 줄 바꿈이 발생하여, '5 and 6'로 시작하는 문단도 눈 그렇게 거슬리지 않기 때문이다. 위의 그 밖의 다른 경우는 그렇지 않다.

- 사람 이름의 surnames와 forename 사이에

  ```
  Donald~E. Knuth                     Juis~I. Trabb~Pardo
  Bartel~Leendert van~der~Waerden     Charles~XII
  ```

  'Don-'과 'ald E. Knuth'가 차라리 'Donald'와 'E. Knuth' 보다 낫다.

- 수식 기호들 사이에

  ```
  dimension~$d$     width~$w$     function~$f(x)$
  string~$s$ of length~$l$
  ```

  하지만 마지막 예가 다음과 같이 사용될 때는 아래 처럼 한다.

  ```
  string~$s$ of length $l$~or more.
  ```

- 시리즈를 나타내는 기호들 사이

```
1,~2, or~3
$a$,~$b$, and~$c$
1,~2, \dots,~$n$.
```

- 어떤 기호가 전치사의 목적어로 사용될 때,

```
of~$x$
from 0 to~1
increase $z$ by~1
in common with~$m$.
```

하지만 전치사의 목적어가 여러 개 일때는 위 법칙이 적용되지 않는다.

```
of $u$~and~$v$
```

- When mathematical phrases are rendered in words:

```
equals~$n$      less than ~$\epsilon$      (given~$X$)
mod~2           modulo~$p^e$               for all large~$n$
```

- case 문이 문단 안에서 번호 붙여서 사용 될 때

```
(b)~Show that $f(x)$ is (1)~continuous; (2)~bounded.
```

3 틸데 사용법 (2)

작은나무 2006-05-02T04:34:29

93쪽: 앞에서 ~ 를 사용하는 방법과 규칙을 살펴보았다. 별로 어렵지 않다. 아무렇지도 않아보이는 ~ 를 넣음으로해서 문서를 보다 온전하게 만들 수 있다.

~는 간격을 만들지만 그 간격에서는 줄바꿈이 일어나지 않는다고 배웠다. 하지만 간격에서 뿐만아니라 하이픈이나 대쉬에서도 줄바꿈이 일어나는 것을 원치 않을 것이다. 이것은 어떻게 해결할까? 하이픈이나 대쉬 다음에 ~ 를 이용한다? 불필요한 간격이 생겨서 안될 것 같다. \hbox를 이용하는 방법이 있다. 텍에게 있어서 박스는 더이상 나눌 수 없는 하나의 단위이다. 다음과 같은 경우이다.

```
Table~\hbox{B-8}
```

먼저 글에서 왜 \hbox가 쓰였나 궁금했을 것이다. \hbox는 페이지의 범위를 나타내는 곳에서도 사용하면 좋다.

```
\hbox{13--22}
```

-- 다음에 줄바꿈이 일어나서 22로 새로 시작하는 줄은 별로 보기 좋지 않을 것이다. ~의 사용법을 익히는 의미에서 연습문제를 풀어보자.

연습문제: 14.1: 어떻게 타이핑하면 아래와 같은 결과물을 얻을 수 있을까?

(cf. Chapter 12).
Chapters 12 and 21.
line 16 of Chapter 6's story
lines 7 to 11
lines 2, 3, 4, and 5.
(2) a big black bar
All 256 characters are initially of category 12,
letter x in family 1.
the factor f, where n is 1000 times f.

```
(cf.~Chapter~12).\cr
Chapters 12 and~21.\cr
line~16 of Chapter~6's {\tt story}\cr
lines 7 to~11\cr
lines 2,~3, 4, and~5.\cr
(2)~a big black bar\cr
All 256~characters are initially of category~12,\cr
letter~{\tt x} in family~1.\cr
the factor~$f$, where $n$~is 1000~times~$f$.\cr
```

연습문제: 14.2: ~ 사용 규칙대로라면 for all n greater than n_0는 어떻게 타이핑 해야할까?

```
for all $n$~greater than~$n_0$
```

연습문제: 14.3: exercise 4.3.2–15 는 어떻게 타이핑 할까?

\hbox를 이용하는 하면 될 것 같다.

```
exercise \hbox{4.3.2--15}
```

exercise 다음에 ~는 필요 없다. 왜냐하면 4.3.2--15이 충분히 길기 때문에 줄이 이것으로 새로 시작한다고 해서 이상할 것이 하나도 없기 때문이다.

연습문제: 14.4: Chapter~12라고 타이핑 하는 것이 \hbox{Chapter 12}라고 하는 것 보다 더 좋은 이유는 무엇일까?

~로 인하여 생긴 간격은 그 줄의 환경에 따라서 줄어들 수도 늘어날 수도 있다. 하지만 \hbox 안의 간격은 고정된 것이어서 신축성이 없다. 그리고 Chapter 에서 하이픈을 이용하여 줄 바꿈이 일어 날 수도 있기때문에 Chapter 12 가 더 나은 방법이다. 위에서 설명했듯이 \hbox는 하나의 단위 이기때문에 줄바꿈이 생길 수 없다.

연습문제: 14.5: 텍은 종종 수식은 등호기호(=) 뒤에서 줄바꿈을 하기도 한다. 그것을 막으려면 어떻게 하면 될까? 예를 들어 '$x = 0$'에서는 줄바꿈이 발생하면 안된다.

> 우선 두가지 방법을 생각할 수 있다. `\hbox{$x=0$}`과 `${x=0}$`이다. 별 문제 없어보이지만, 바로 위의 문제에서 본 바와 같이 등호기호(=) 좌우의 간격이 고정된다는 것이 문제다. 이를 해결하는 멋진 방법이 있다. `$x=\bs{}nobreak 0$` \nobreak에 대해서는 나중에 언급할 기회가 있겠다.

연습문제: 14.6: 우리가 직접 입력하는 하이픈이나 대쉬 다음에 줄 바꿈을 하지 말라고 텍에게 어떻게 부탁하면 될까?

> 앞으로 배울 명령어 `\exhyphenpenalty=10000`를 이용하면 하이픈이나 대쉬 다음에서의 줄바꿈을 막을 수 있다. 비슷하게 `\hyphenpenalty=10000`라고 하면, 우리가 직접 입력하는 하이픈이나 대쉬가 아닌 단어 내에서 줄바꿈으로 발생하는 discretionary 하이픈에서도 줄바꿈을 막을 수 있다. 즉 줄바꿈을 위해서 단어내에서 하이픈이 생기지 않는다.

이상으로 ~ 의 사용법을 알아보았다. 어렵지 않다. 몇가지 규칙이 있기는 하지만, 외울 필요는 없어 보인다. 몇번 사용하다보면 저절로 알 수 있을것 같다. 모두다 상식적으로 그럴싸한 규칙이므로. `:)`

4 \obeylines

작은나무 2006-05-02T14:38:25

94쪽: 문단 중간에서 강제로 줄을 바꾸고자 한다면 어떻게 하면 될까? 간단하다. 단순히 '\break' 라고만 하면 된다. 하지만 그렇게하면, 텍은 어떻게 해서든지 양끝을 동일하게 맞추려고 하기때문에 단어들 사이에 엄청난 간격이 생기고 만다. 예를 들면 다음과 같다.

```
양끝을 동일하게 맞추려고 하기때문에 단어들 사이에 엄청난 간격이 생기고 만다.
예를 들면 다음과 같다.\break
```

양끝을 동일하게 맞추려고 하기때문에 단어들 사이에 엄청난 간격이 생기고 만다. 예
를 들면 다음과 같다.

이를 방지하려면, '`\hfil\break`'라고 한다. 이는 줄을 바꿀 때, 나머지를 공백으로 모두 채우고, 줄을 바꾼다. 그런데, 이경우는 새로 시작하는 문단은 들여쓰기가 안된다. 다음과 같다.

```
양끝을 동일하게 맞추려고 하기때문에 단어들 사이에 엄청난 간격이 생기고 만다.
예를 들면 다음과 같다.\hfil\break
이는 줄을 바꿀 때, 나머지를 공백으로 모두 채우고, 줄을 바꾼다.
```

양끝을 동일하게 맞추려고 하기때문에 단어들 사이에 엄청난 간격이 생기고 만다.
예를 들면 다음과 같다.
이는 줄을 바꿀 때, 나머지를 공백으로 모두 채우고, 줄을 바꾼다.

줄 바꿈과 관련하여 재미있는 예를 들어보자. 다음은 어떻게 입력하여 얻은 결과일까?

나 보기가 역겨워
  가실 때에는
말없이 고이 보내 드리오리다.

영변에 약산
  진달래꽃
아름 따다 가실 길에 뿌리오리다.

먼저 생각 할 수 있는 방법은 모든 줄의 마지막에 '\par'를 넣으면 될것 같다. 맞다. 하지만 줄의 마지막에 매번 '\par'를 넣은 것은 곤역이겠다. 이런 경우를 대비해서 플레인텍은 친절하게도 '\obeylines' 라는 매크로를 제공한다. 어떤 문단의 처음에 \obeylines 라고 하고, 그 문단을 그룹으로 만들면, 그 문단의 매 줄의 끝마다 \par를 넣을 것과 같은 효과를 낸다. 물론 그 줄의 마지막에 %를 넣거나, 줄을 매우 길게 하지 않는다면 말이다. 그래서 위의 시는 다음과 같은 입력으로 얻은 결과이다.

```
{\obeylines\smallskip
나 보기가 역겨워
\quad가실 때에는
말없이 고이 보내 드리오리다.
\bigskip
영변에 약산
\quad진달래꽃
아름 따다 가실 길에 뿌리오리다.
\smallskip}
```

여기서 \obeylines를 적용하고자 하는 곳을 그룹으로 만드는 것에 주목한다. 그렇지 않으면, 입력 파일을 끝까지 모두 시쓰기 모드가 될 것이다. :)

연습문제: 14.7: 위의 시에서 \quad가 하는 일을 설명해보라. \quad를 \indent로 바꾸면 어떻게 될까?

> \quad는 대문자 M의 너비만큼 \hskip 하라는 뜻이므로 그 너비만큼 들여쓰기 하는 효과가 있다. 들여쓰기라면 \indent라는 준비된 primitive 명령어가 있지 않는가? 왜 들여쓰기를 위해서 \indent를 사용하지 않고 \quad를 사용했을까? 왜냐하면 \indent를 사용하면 들여쓰기가 되지 않기 때문이다. 문단을 시작할때는 \indent를 명시적으로 쓰지 않아도 텍은 암묵적으로 \indent가 있다고 생각하고 들여쓰기를 한다. 즉, 문단을 시작할때는 들여쓰기를 위해서 \indent를 써도 되고 안써도 된다는 것이다. 둘다 마찬가지이다. 따라서 \indent를 이용해서 들여쓰기를 하려면, '\indent\indent' 로 하면 된다.

5 줄바꿈 알고리듬 대충 살펴보면 이렇다.

작은나무 2006-05-03T14:29:01

94쪽: 텍의 줄 바꿈 알고리듬, 그까이꺼 대~~~충 이렇다. 줄을 바꾸는 위치 (breakpoints)는 단어들 사이나 하이픈 다음이 되고, 그로 인해 생긴 줄들의 badness는 주어진 \tolerance를 넘지 않도록 한다는 것이 기본 원칙이다. \tolerance를 넘기지 않고는 도저히 줄바꿈을 할 수 없을 때는 overfull 박스를 만들어낸다. \tolerance를 넘기지 않고 줄바꿈을 할 수 있다면, 수학적으로 최적의 방법으로 줄바꿈 위치를 찾아낸다. 주어진 문단에서 여러 개의 줄바꿈 위치를 구해 냈을때 그들이 만들어내는 줄들은 최소한의 결점(demerits)을 갖도록 하는것이지라.

사실 위에서 대충 알아본 줄바꿈 알고리듬은 너무 드문드문 알아본 것이다. 보다 자세히 알아보기 전에 사전 지식을 갖추자. 한 문단이 여러 개의 줄들로 나뉘기 전에, 그 문단은 사실 텍 내부에서는 "수평리스트(horizontal list)"로 취급된다, 즉, 텍이 수평모드(horizontal mode) 일 때에 긁어 모은 아이템들의 나열이라는 것이다. 그동안 편하게 수평리스트라는 것은 박스(box)들과 글루(glue)로 구성된다고 말해왔다. 이 말이 거짓말은 아니었지만, 좀 무성의한 대답이었다. 진실은 이렇다. 수평리스트를 구성하는 아이템들은 다음 중의 하나이다.

- 박스 (한 문자 혹은 ligature 또는 선 또는 hbox 또는 vbox)
- a discretionary break (이에 대해서는 곧 알아볼 것이다)
- a "whatsit" (나중에 설명할 좀 특별한 것)
- vertical material (\mark 또는 \vadjust 또는 \insert 로 부터의)
- a glob of glue (또는 \leaders)
- a kern (신축성이 없는 글루)
- a penalty (줄바꿈 위치의 바람직하지 않은 정도)
- "math-on"(수식의 시작) 또는 "math-off"(수식의 끝)

마지막 네 종류(glue, kern, penalty, math items)는, 줄바꿈 시점에 변경되거나 사라질 수 있는 것들이어서, "폐기될 수 있는(discardable)"이라고 불린다. 이를 제외한 첫번째 네 종류들은 없어질 수 있는 것들이 아니고 언제나 그 형태 그대로 유지하므로 폐기될 수 없는(nondiscardable)이라고 불린다.

위의 수평리스트들의 아이템에는 어떤 것들이 있고, 그 아이템들은 discardable 한지 아닌지는 알아 둘 필요가 있다. 나중에 요긴하게 쓰일 것이기 때문이다. :)

6 discretionary break

작은나무 2006-05-07T03:40:05

95–96쪽: 먼저 discretionary의 사전적 의미를 알아보자.

> dis-cre-tion-ar-y
> a. 《문어》 임의의, 자유재량의
> ¶ a discretionary order 【상업】 중매인에게 시세대로 매매를 일임하는 주문
> ¶ a discretionary principle 독단주의
> discretionary powers to act 자유로이 행동할 수 있는 재량권

discretionary break를 우리말로 뭐라고 해야할지 모르겠으나, 영어에서 단어 중간에 줄바꿈이 발생할 경우에 사용된다. discretionary break는 다음과 같은 세 가지 텍스트로 이루어진다.

- “pre-break” 텍스트
- “post-break” 텍스트
- “no-break” 텍스트

discretionary break가 사용되는 방법은 이렇다. 이 discretionary break에서 줄바꿈이 일어날 경우, 현재의 줄 마지막에 pre-break 텍스트가 나오고, 줄바꿈된 다음 줄의 맨 처음에 post-break 텍스트가 나오고, 줄발꿈이 발생하지 않으면, 현재의 줄에 no-break 텍스트가 나온다.

```
\discretionary{<pre-break 텍스트>}{<post-break 텍스트>}{<no-break 텍스트>}
```

예를 들어보자. 텍이 difficult 단어에서 두 개의 f 사이에서 하이픈을 발생시키게 하려면, 다음과 같이 하면 된다.

```
di\discretionary{f-}{fi}{ffi}cult.
```

다행인 점은 모든 단어마다 위와 같이 하지 않아도 된다는 것이다: 텍의 그 유명한 하이프테이션 알고리듬이 우리 모르게 뒤에서 다 알아서 해준다고 한다. 더 다행인 점은 우리글 한글에는 전혀 해당사항 없음 이다. :)

discretionary break가 가장 일반적으로 쓰이는 경우는

```
\discretionary{-}{}{}
```

인데, 이는 너무 자주 사용되어서 텍에 \-와 같은 control symbol로 대신하고 있다. 그래서 위의 difficult를 예를 들면 이렇다. `dif\-ficult`. 앞서 설명대로 텍은 자동적으로 하이픈을 해주지만, 우리가 discretionary break를 이용해서 어떻게 하이프네이션을 하라고 정해놓으면, 텍은 그 단어에 대해서는 우리 의견을 존중한다.

연습문제: 14.8: 독일어에는 하이픈할때 특이한 경우가 있다. ‘backen’이 하이픈 될때는 ‘bak-ken’으로 c 가 k로 바뀌고, ‘Bettuch’가 하이픈 될때는 ‘Bett-tch’가 된다. 이는 어떻게 하면 될까?

> 위의 단어 뿐만아니라 ‘ck’, ‘tt’가 들어가는 모든 독일어 단어에 해당되므로 아예 매크로를 만들면 좋을 것 같다.
>
> ```
> \def\ck/{\discretionary{k-}{k}{ck}}
> \def\ttt/{\discretionary{-}{t}{}}
> ```

다음 글부터 본격적인 줄바꿈 알고리듬에 들어갑니다.

7 줄바꿈이 일어나는 곳: 다섯가지 경우

작은나무 2006-05-12T22:35:16

96쪽: 줄바꿈 알고리듬을 알아보기 전에 줄바꿈이 어디에서 발생하는지를 상식적으로 생각해보자. 어디일까? 당연히 단어들 사이일 것이다. 그리고 단어들 사이의 간격을 적절히 보기 좋게 유지하면서 단어들 사이에서 줄바꿈이 불가능하다면, 어쩔 수 없이 앞에서 설명한 discretionary break를 사용해서 단어 중간에서 하이픈을 이용해서 나누고 그 하이픈 다음에서 줄바꿈이 일어날 것이다.

즉, 정리하면 단어들 사이에서 줄바꿈이 일어나면 더이상 바랄 것이 없고, 그게 여의치 않다면 하이픈을 이용한다는 것이다. 따라서 텍은 가능하면 하이픈을 사용하지 않고 줄바꿈을 할 수 있을지 알아볼 것이다. 텍은 한 문단을 줄들로 나눌때 두단계(2 pass)에 걸쳐서 시도한다. 먼저 하이픈이 없다고 하고 줄들로 나누려고 시도하고, 그게 안되면 조건을 조금 완화해서 하이픈을 허용해서 주어진 문단을 줄들로 나누려고 한다. 이때 사용되는 텍의 명령어들이 \pretolerance, \tolerance 등이다. 이에 대한 친절하고 자세한 설명은 다음을 참고한다. 꼭 읽어보라.

- badness and tolerance[5)]

읽었다고 가정하고 그냥 넘어간다. :) 그래도 한번 정리해보자. 플레인텍은 기본 값으로 다음과 같이 정한다.

- `\pretolerance=100`
- `\tolerance=200`

만약 `\pretolerance=10000`이면 무슨 일이 벌어질까? 그렇다. 첫번째 단계에서 무조건 성공이다. 텍은 주어진 단어들을 하이프네이션 하려고 시도도 하지 않는다는 말이다. 즉, 두번째 단계까지 갈것 없이 첫번째 단계에서 모든 것을 끝내겠다는 말이다. 그래서 줄들의 간격이 간혹 보기 흉하게 넓게 혹은 좁게 되기도 한다.

`\pretolerance=-1`이라면? 첫번째 단계는 무조건 실패이므로, 첫번째 단계를 그냥 통과하고, 두번째 단계 즉, 언제든지 하이프이 발생할 수 있다고 가정하고 줄들을 만들어 나갈 것이다.

우리는 앞에서 줄바꿈은 단어들 사이나 하이픈 다음에서 일어난다고 들어서 대충 알고 있다. 정확히 하면 다음과 같다. 즉 줄바꿈은 실제로 다음의 다섯가지 경우에서 일어난다.

- glue. 이 글루는 수평리스트를 구성하는 아이템들 중 non-discardable 아이템들 다음에 오는 글루이어야 하고, 수식에서 사용되는 글루가 아니어야 한다. 이경우, 즉 글루에서 줄바꿈이 일어날 때는 이 글루의 왼쪽끝에서 일어난다. 예를 들어 이 글루가 1cm이라고 하면 1cm글루가 시작하기 바로 전에 줄바꿈이 일어난다는 말이다.
- kern. 이 kern다음에 바로 글루가 나와야 하고, 수식에서 사용되는 kern이 아니어야 한다.
- math-off. 이 math-off 다음에 바로 글루가 나와야 한다.

---

5) http://faq.ktug.or.kr/faq/LittleTree/ReadingTeXbook/2006-03#90cecc9f62df8b348dc17a6cbb3df42f

- penalty.
- discretionary break.

두 개의 글루가 연이어 나온다면, 두번째 글루에서는 줄바꿈이 일어나지 않는다. 왜냐? 그 두번째 글루 앞에 있는 것 또한 글루인데, 수평리스트를 구성하는 아이템에서 글루는 discardable 하다고 했고, 방금전 설명에서는 글루에서 줄바꿈이 일어나려면, 그 글루는 non-discardable 아이템 다음에 오는 글루이어야 한다고 했다.

다음 글에서는 월드컵도 다가오고 하니, 페널티(킥)에 대해서 알아본다. **penalty.**

▶ *"12장 글루, 13장 모드"를 건너뛰고 14장 내용을 다루고있으니, 조금 많이 거시기합니다. 양심(?)에 걸리는 면도 있고... 왜냐하면, 12장에서 자세히 설명하는 glue, badness, 13장에서 설명하는 수평, 수직, 수식 모드... 이런 것들에 대한 개념이 충분해야 이 글을 읽는 분들이 14장을 온전히 이해하면서 읽어나갈 수 있을텐데 말입니다. (그런데, 이 글을 읽는 분들은 몇 분 안되는 것 같고, 그 분들은 앞에 언급한 것들에 대한 기본 개념들을 이 글을 쓰고 있는 저보다 훨씬 더 정확히 파악하고 계실테니, 쓸데 없는 걱정인 듯합니다. :)*
아무래도 12, 13장을 다루는 시간을 곧 가져야 할 듯..... 쩝. 하긴 12, 13장을 다루다보면, 그 앞장들을 또 언급해야 할지도 모르겠습니다. ㅎㅎㅎ 이것참... 그럼 아예 처음부터? – [작은나무] [[DateTime(2006-05-12T22:55:33)]]

▶ *제 경험에 의하면 뜻밖에 글을 읽고 계신, 그러나 말씀은 없는 분들이 많습니다. 이를테면 이런 거죠... 나중에 이거 한번 봐야겠다... 제가 예전에 썼던 악보 조판 관련 글을 조인성 교수께서 10년동안이나 기억하고 계셨다는 말씀도 하셨자나요. 아무튼 제가 바라는 것은 중간중간 중간점검겸 요약반쳬본 형식의 pdf가 가끔 제공되면 참 좋겠다... 그런 생각을 하지요. :) 좋은 글 감사합니다. – [Karnes] [[DateTime(2006-05-13T04:20:55)]]*

8 penalty revisited

작은나무 2006-05-14T11:09:10

96–97쪽: 사실 penalty에 대해서는 penalty[6]에서 이미 다루었다. 한 두달여 전에 처음으로 페널티를 접하면서 쓴 글인데, 지금 다시 읽어봐도 크게 잘못 쓴게 없는 것 같다. :) 꼭 읽어보기 바란다. 여기서는 그때 언급하지 않은 내용 위주로 설명한다.

"페널티", 어느 지점에서 줄바꿈 했을때, 그 지점이 줄바꿈을 하기에 적합한 곳인가에 대한 기준으로, 부적합한 정도에 따라서 그 지점은 페널티를 문다. 줄바꾸는 지점을 영어로는 breakpoint 라고 한다. 다시 상식적으로 생각해보자. 줄바꿈을 하는 가장 이상적인 위치는 즉 가장 좋은 브레이크뽀인뜨는 단어들 사이의 간격에서 이다. 그러한 간격에서 줄바꿈을 하는데, 페널티를 물릴 수는 없는 노릇이다. 또, 단어 내부에서는 가급적이면 줄바꿈이 일어나지 말아야 하는데, 어쩔 수 없이 일어나야 한다면, 당근 페널티를 물려야 한다.

정리하자. 앞의 글에서 줄바꿈이 일어나는 다섯 가지 경우를 살펴보았다. 그 각각의 경우에 대해서 페널티를 물리자. :) 먼저 처음 세가지 경우, glue, kern, math-off는 방금 전의 설명에서 간격에 해당하므로 페널티를 물지 않는다. 페널티를 물지 않는다는 소리는 페널티가 0이라는 뜻이다.

---

6) http://faq.ktug.or.kr/faq/LittleTree/ReadingTeXbook/2006-03#136ecb1ee6db1d00f4a7e2395f6db97b

네번째 경우, penalty, 네번째 경우는 페널티를 명시적으로 정하는 것이다. 즉 자진납세에 해당한다. 나는 어느정도의 페널티를 물고서라도 이 지점에서 줄바꿈을 하겠다는 의지의 표현이다. 마지막 다섯번째 경우, discretionary break, 이 경우에 플레인텍은 \hyphenpenalty, \exhyphenpenalty라는 프리미티브의 기본 값을 가지고 있다. 이 두 프리미티브는 보기에는 아주 어려운 개념처럼 보이는데, 전혀 그렇지 않다. discretionary break는 pre-break, post-break, nobreak의 세가지 텍스트로 이루어진다고 알고있다. \hyphenpenlaty는 그 단어에서 의미를 유추할 수 있듯이, 단어가 하이픈을 이용해서 나뉘어 짐으로 줄바꿈이 됐을때, pre-break 텍스트가 nonempty일때, 물어야 하는 페널티이고, \exhyphenpenalty는 pre-break 텍스트가 empty일때, 물어야하는 페널티이다. 플레인텍은 기본값으로 이 두 프리미티브에 50을 부여한다. `\hyphenpenalty=50`, `\exhyphenpenalty=50`.

페널티의 값이 음수라면? 음수의 페널티는 보너스이다. 그렇지 않은가? 어떤 브레이크포인트의 페널티가 -100이라고 하자. 텍은 그 지점에서 줄바꿈을 하면 100이라는 보너스가 주어지므로 줄바꿈하려고 꽤나 노력할 것이다. 그리고 그러한 지점에서 줄바꿈이 발생해서 생긴 줄은 보너스를 얻은 줄이기 때문에 어떤 메리트(merit)를 가지고 있을 것이다. 줄바꿈 알고리듬의 궁극적인 목적은 가능한한 모든 줄들이 메리트를 최대한 갖도록 즉, 디메리트(demerit, 단점)를 최소화하도록 하는데 있다. (나중에 복잡한 수식과 자세히 알아볼 것이다.)

아마도. 마지막으로 `\penalty10000`은 무슨 뜻인가? 10000이라는 숫자는 텍에서는 무한대를 나타내므로, 페널티를 무한대로 물어야 한다. 즉 절대로 줄바꿈을 하지 말라는 소리다. 이러한 뜻으로 플레인텍은 `\nobreak`라는 매크로를 가지고 있다. 이제 ~의 뜻을 정확히 알 수 있다. 바로 `\nobreak\␣`이다. :)

응용문제 들어간다.

연습문제: 14.9: 플레인텍의 매크로 \break는 어떻게 정의되어 있을까?

```
\def\break{\penalty-10000} % 너무 쉽다.
```

연습문제: 14.10: \nobreak\break 또는 \break\nobreak 라고 하면 어떤 일이 벌어질까?

마치 `\nobreak`가 없는양, `\break`만 적용되서 줄바꿈이 발생한다. 왜냐하면 `\break`는 또다른 페널티에 의해서 취소될 수 없기 때문이다. `\break`는 텍에게 있어서 반드시 줄바꿈을 해야하는 최상의 브레이크뽀인뜨 인것이다. (best candidit for line breaking)일반적으로 한 줄에 두개의 페널티가 있다면, 그 둘이 만들어내는 효과는 그 둘 중 값이 작은 페널티 하나만 주어진 것과 동일하다. 단, 그 두 페널티는 `-10000`이 아니어야 한다.

9 demerits revisited

작은나무 2006-05-15T13:54:08

일단 줄바꿈이 발생하면, 텍은 그 브레이크포인트 다음에 나오는 모든 discardable 아이템들을 제거 한다. 언제까지? non-discardable 아이템 혹은 새로운 브레이크포이트 까지. 예를들어, 글루나 페날티 여러 개가 연달아 나오면 그들을 마치 하나의 글루 혹은 페널티로 취급한다. 그리고

수식의 시작 및 끝은 \mathsurround에 정해진 값에 의해서 간격을 가질 수 있는데, 그 간격들은 수식이 문단의 제일 앞에 나오면 수식 시작에 나오는 간격이, 문단의 제일 마지막에 나오면, 수식의 마지막에 나오는 간격은 없어진다. 이는 당연한 것 같다.

주어진 줄에 있는 단어들 사이의 간격의 보기 좋고 나쁨을 나타내는 badness를 보다 수학적으로 알아보자. 줄의 badness는 정수 값인데, 그 줄에 있는 글루들이 늘어나거나 줄어드는 비율의 세제곱의 대략 100배이다. 무슨 말인지 예를들어 설명하면, 어떤 한 줄이 있고, 그 줄에 들어있는 글루들이 줄어들 수 있는 한계가 10이고, 실제로 9만큼 줄어들었다면, 그 줄의 badness는 $73(100\times(9/10)^3 = 72.9)$이다. 같은 맥락으로, 어떤 줄의 글루가 늘어날 수 있는 한계보다 두배만큼 늘어나면 그 줄의 badness는 $800(100\times2^3)$이 된다. 그런데, 이와 같은 수식 계산으로 badness를 구할 때 그 값이 10000을 넘을 때가 있다. 이 경우의 badness는 10000으로 한다. Overfull 박스들은 badness가 10000인 것으로 간주되서, 가능하면 그러한 박스들이 생기지 않도록 해야 한다.

어떤 줄의 badness가 13이상일때, 그 줄을 "tight" 혹은 "loose" 하다고 하고, 100이상일때 "very loose" 하다고 한다. 12이하일때는 "decent"라고 한다. 그리고 tight한 줄 다음에 loose 나 very loose 한 줄이 나오거나, decent한 줄 다음에 very loose한 줄이 나오면, 그 이웃한 줄들을 "visually incompatible" 하다고 한다.

이제 줄바꿈 알고리듬이 거의 그 실체를 드러낼때가 되었다. 텍은 줄바꿈 지점들의 집합을 여러개 가지고 있고, 그 집합들을 예비 브레이크포인트 집합이라고 하고, 예비 브레이크포인트 집함에서 각 브레이크포이트가 가지는 디메리트(demerit)의 합이 최소가 되는 브레이크포인트 집합을 선택 하는 것이 바로 줄바꿈 알고리듬이다. 메리트의 개념에 대해서는 앞의 글에서 살펴보았다. 어떤 브레이크포인트가 보너스를 받으며 (음수의 페널티를 가지면) 줄바꿈을 하면 그로 인해 생긴 줄들은 보기가 좋고 그로 인해서 메리트를 갖는다고 했다.

(수식들어갑니다. 긴장하세요.)

어떤 줄이 주어지고, 그 줄의 badness를 b라고 하고, 그 줄의 브레이크포인트의 페널티를 p라고하자. 일단 p가 10000 이상이거나 b가 주어진 tolerance나 pretolerance를 초과하지 않는다고 가정하자. 이러한 경우는 텍이 가지고 있는 알고리듬으로 처리할 수 없는 경우이다. 그러면, 각 줄의 디메리트를 구하는 공식은 다음과 같다.

$$d = \begin{cases} (l+b)^2 + p^2, & \text{if } 0 \le p < 10000; \\ (l+b)^2 - p^2, & \text{if } -10000 < p < 0; \\ (l+b)^2, & \text{if } p \le -10000. \end{cases}$$

여기서 l은 현재의 \linepenalty의 값이데, 이 값은 주어진 문단의 모든 줄의 baddness에 더해지는 값이다. 플레이텍은 \linepenalty=10으로 정한다. 디메리트 구하는 것을 예로들어보자. 어떤 줄의 badness를 20이고, 그 줄은 글루로 끝난다고 하자. 그러면 플레인텍은 그 줄의 디메리트를 $(10+20)^2 = 900$ 이라고 정할 것이다. 왜냐하면 글루에서 줄바꿈이 일어나므로, 페널티는 0이므로 위의 수식에서 첫번째에 해당한다. 그리고 플레인텍은 \linepenalty의 값을 기본으로 10으로 갖고있기때문에. 위의 수식에 의하면, 주어진 문단의 디메리트들의 총합을 최소화 하는 것은 대충 badness와 페널티들의 제곱의 합을 최소화 하는 것과 같다. 이 말은 다시 각 줄에 대해서 badness를 최소화 하는 것과 같은 말이다!!!!!!!!!!!!!

연습문제: 14.11: 위의 디메리트 구하는 공식을 잘 살펴보면, p의 값에 따라서 세개의 구간으로 나뉘어져 있는데, p의 값이 -10000 인 경우에 함수가 연속이지 않는다. 그 지점에서 함수가 연속이려면 (매끄럽게 이어지려면) 마지막 p가 -10000 이하일 경우 수식은 $(l+b)^2 - 10000^2$ 이어야 할 것이다. 하지만, 그렇지 않다. 이것은 무엇때문일까? :)

> 이미 알고있듯이 페널티가 -10000 이하이면 무조건 줄바꿈이다. 따라서 다른 브레이크포인트가 어찌됐던간에 상관없이 그 지점에서는 무조건 줄바꿈이다. 즉 브레이크포인트 하나는 이미 결정된 셈이다. 따라서 10000^2(= 100,000,000) 1억이나 되는 큰 상수를 뺄이유가 없고, 만약 뺀다면 전체 디메리트의 총합에 영향을 미치게 되어서 나머지 브레이크포인트를 구하는데 있어서 장애가 된다. 더구나 1억을 빼면 컴퓨터 계산상에도 오버플로우가 발생할지도 모른다. :)

휴~~~ 이제 줄바꿈 알고리듬(줄들의 디메리트의 총합을 최소화하는 것)의 최대 관문인 디메리트를 계산하는 것을 알아보았다. 거의 다 끝났다. 이렇게 나온 디메리트에 그 줄이 처한 환경에 따라서 몇가지 값을을 더하거나 빼주는 작업만이 남았다. 만약 이웃하는 두 줄이 얼마전에 설명한 visually incompatible 하다면, `\adjdemerits`의 값은 이미 구한 디메리트 d에 더한다. 그리고, 이웃한 두줄이 discretionary break로 끝나면 `\doublehyphendemerits`를 d에 더한다. 끝으로 문단에서 마지막 두번째 줄이 discretionary로 끝나면, `\finalhyphendemerits`를 d에 더한다. 플레인텍에서 이러한 값들의 기본값은 `\adjdemerits=10000`, `\doublehyphendemerits=10000`, `\finalhyphendemerits=5000` 이다. badness나 tolerance, penalty의 값들은 대개 그 단위가 비슷하다. 하지만, 방금전에 알아본 디메리트에 관련된 숫자들은 엄청나게 크다. 수식에서도 알수 있듯이 badness의 제곱으로 디메리트를 구하기 때문이다. `:)` 진짜 끝이다.

10 줄바꿈 알고리듬의 백미

작은나무 2006-05-16T01:23:59

98–99쪽: 이번 글은 현재 다루고 있는 텍의 줄바꿈 알고리듬의 백미라고 할 수 있다. 우리들이 직접 텍이되어서 주어진 문단을 지금까지 배운 알고리듬을 토대로 수학문제 풀듯이 눈으로 직접보고 손으로 직접 풀어서 줄들을 만들어내는 것이다. 그러기 위해서는 여러분이 작성하는 텍 입력파일에 `\tracingparagraphs=1`이라고 지정해야한다. 그러면 텍은 log 파일에 줄을 만드면서 한 모든 짓거리들을 상세히 보여줄 것이다. 우리가 할 일은 그 로그파일을 보고 즐거워하면 된다. `:)`

예를 들어 `story.tex`이라는 파일을 하나 만든다.

```
\tracingparagraphs=1
\hsize=2.5in
\tolerance=1000
\hrule
\vskip 1in
\centerline{\bf A SHORT STORY}
\vskip 6pt
```

```
\centerline{\sl    by A. U. Thor} % !‘?‘?! (modified)
\vskip .5cm
Once upon a time, in a distant
  galaxy called \"O\"o\c c,
there lived a computer
named R.~J. Drofnats.

Mr.~Drofnats---or ‘‘R. J.,’’ as
he preferred to be called---% error has been fixed!
was happiest when he was at work
typesetting beautiful documents.
\vskip 1in
\hrule
\vfill\eject
```

위 파일에서 주목할 것은 텍스트의 폭은 2.5인치로 매우 작고, 텍의 인내력을 1000으로 정했다는 것이다. 이는 텍의 기본 인내력 200의 5배나되는 큰 값이다. 위 입력파일을 가지고 컴파일을 하고 story.log 파일을 살펴보자. 여기서는 두번째 문단에 해당하는 로그 파일을 살펴 볼 것이다.

```
[]\tenrm Mr. Drofnats---or ‘‘R. J.,’’ as he pre-
@\discretionary via @@0 b=0 p=50 d=2600
@@1: line 1.2- t=2600 -> @@0
ferred to be called---was hap-pi-est when
@ via @@1 b=131 p=0 d=29881
@@2: line 2.0 t=32481 -> @@1
he
@ via @@1 b=25 p=0 d=1225
@@3: line 2.3 t=3825 -> @@1
was at work type-set-ting beau-ti-ful doc-
@\discretionary via @@2 b=1 p=50 d=12621
@\discretionary via @@3 b=291 p=50 d=103101
@@4: line 3.2- t=45102 -> @@2
u-
@\discretionary via @@3 b=44 p=50 d=15416
@@5: line 3.1- t=19241 -> @@3
ments.
@\par via @@4 b=0 p=-10000 d=5100
@\par via @@5 b=0 p=-10000 d=5100
@@6: line 4.2- t=24341 -> @@5
```

(장난하냐? 즐기면 된다며? 백미라며?) 여러분의 원성이 들리는것 같다. 원래 작은나무가 오바가 좀 심하다. 하지만 정말 신기하고 재미있는 것을 어떻하랴. 암호를 해독해보자. 암호해독의 기본 지식은 여태까지 우리가 배워온 discretionary break, badness, 각종 penalty들, demerit 등이다. 복잡해 보이지만 전혀 그렇지 않다. 작은나무를 믿고 한번 해보자. :)

우선 표기법부터 살펴본다.

- 위 로그 파일을 보면 @@로 시작하는 줄들이 있다. 세어보니 여섯 줄이 있고, @@ 다음에는 친절하게도 번호가 순차적으로 붙어있어서 맨마지막 @@를 보면 전체 @@가 몇개 있는

지 알 수 있다. @@로 시작하는 것들이 바로 가능한 브레이크뽀인뜨(feasible breakpoints)이다. story.tex에서 `Mr.~Drofnats`로 시작하는 두번째 문단은 여섯 개의 줄바꿈지점을 가질 수 있다는 뜻이다. 문단의 시작 또한 브레이크포인트라고 볼 수 있기에 문단의 시작은 @@0으로 나타낸다. 결국은 텍이 해야할 일은 총 여섯 개의 브레이크포인트에서 최상 조합의 브레이크포인트들을 고르는 것이다. 나중에 최종 출력물을 확인해 볼 것인데, 그것을 보면 줄바꿈이 모두 네번 발생한 것을 알 수 있을 것이다. 즉 가능한 여섯 개에서 최상의 조합 네개를 고르는 것이다. 최상의 조합이라함은? 그렇다. 네개의 디메리트들의 총합이 최소가 되는 조합이다. `:)` @@로 시작하는 것들이 바로 가능한 브레이크뽀인뜨라고 했는데, 이들은 어떻게 구했을까? 텍은 주어진 폭 2.5인치에 가까운 줄들을 만들어 내면서 그 만들어낸 줄들의 badness가 주어진 tolerance(여기서는 1000) 넘지 않는 지점을 모두 골라서 얻은 것들이다.

- 또한 @@가 아닌 @로 시작하는 줄들도 있다.그 줄들이 의미하는 바는 방금전에 설명한 @@로 시작하는 가능한 줄바꿈지점을 얻을 수 있는 방법을 소개하는 것이다. 무슨 뜻인고 하니, 두번째 가능한 브레이크포인트 @@2를 얻을 수 있는 방법은 한가지가 있고, @@4를 얻을 수 있는 방법은, @@4 앞에 @로 시작하는 줄이 두개 있으므로, 두가지가 있는 것이다. 즉 이런것이다. @@1에서 @@2로 가는 길, @@3으로 가는 길이 각각 1가지 있고, @@4로 가는 길은 @@2에서 가는 길과 @@3에서 가는 길, 두가지 있다. @@1에서 @@4로 가는 길은 총 두가지가 있는 셈인데, 텍은 그 중에서 디메리트가 더 작은 길을 택하는 것이다.

- 지금까지의 설명을 정리하는 겸 예를 들어보자. 위 로그 파일에서 `... hap-pi-est when`와 `he` 사이에 `@@2: line 2.0 t=32481 -> @@1`가 있는 것을 볼 수 있다. 즉 두번째 가능한 브레이크포인트는 단어 "when"과 "he" 사이의 간격에서 줄바꿈이 일어날 수 있다는 것이다. "line 2.0"이 의미하는 바는 이 브레이크포인트는 두번째 줄 끝에서 발생했고, 그로 인해 생긴 줄은 very loose 하다는 것이다. (2.0에서 .0 외에 .1, .2, .3등이 올 수 있는데 각각은 very loose, loose, decent, tight를 의미한다. 이 단어들이 뜻하는 것은 전 글에서 알아본 바와 같이 badness를 기준으로 해서 나온 것들이다. 이경우 이 브레이크포이트로 인해서 생긴 줄의 badness는 `@ via @@1 b=131 p=0 d=29881` 에서 확인 할 수 있듯이 b=131 즉 131인데, badness가 100이상이면 그 줄은 very loose 하다고 했다. 그래서 2.0인 것이다.)

- `line 2.0`과 달리 `line 1.2-`와 같이 끝에 짝대기(하이픈)가 붙은 것들이 있다. 이런 것들은 그 브레이크포인트로 생긴 줄은 discretionary break가 되어서 하이픈으로 끝났다는 것을 의미한다. 실제로 확인해보면, 첫번째 줄은 `...as he pre-`로 끝난 것을 확인 할 수 있다. 그리고, .2이므로 이 줄은 decent 하다.

- 다음으로 알아볼 표기법은 `t=32481`이다. 이것이 의미하는 것은 문단 처음에서부터 시작하여 두번째줄까지의 디메리트의 총합이 32481이라는 소리다. 그리고,

- `-> @@1`이 의미하는 바는 @@2로 도달하는 최상의 길은 @@1을 거쳐서라는 뜻이다. @@2의 경우에 도달 할 수 있는 방법은 선택의 여지가 없이 @@1 하나 밖에 없지만, 예를들어 @@10에 도달할 수 있는 방법은 @@8에서 @@10으로 갈 수도 있고, @@9에서 @@10으로 갈수도 있다. (나중에 확실히 이해할 수 있을 것이다.)

- @@1에서 @@2로 가는 과정을 자세히 살펴보자. 먼저 badness는 131이고, "when"과 "he" 사이의 간격에서 줄바꿈이 일어났으므로 penalty는 0 (p=0) 따라서 디메리트 구하는 공식에 넣어보면 d=29881을 얻을 수 있고(사실은 얻을 수 없다. 연습문제로 확인한다. :)), @@1의 디메리트는 2600이므로 @@2까지의 디메리트의 총합 t=2600+29881=32481이 나온다. (어느 개그프로에서 "조사하면 다나와" 이듯이 계산하면 다나온다.)

- 여기서 아주 재미있다. @@3을 잘 살펴보자. 이 브레이크포인트는 어디서 발생하는가? 로그파일을 조사해보면, "he"와 "was"사이에서 발생한다. @@2와 한 단어(he)차이이다. 이게 무슨 뜻인가? @@2와 @@3는 바로 경쟁자 입장이다. @@1에서 @@4로 가능 방법이 @@1->@@2->@@4 와 @@1->@@3->@@4 으로 가는 방법 두 가지가 있고, 텍은 디메리트가 최소화되는 길을 선택해야 한다. (이제, 가능한 브레이크포인트가 무엇이고, 그들의 최상의 조합이 무엇인지 이해할 것이다.) 텍은 과연 위 두가지 조합중 어떤 조합을 택할 것인가? 로그 파일을 보자. 앞의 설명에서 @로 시작하는 줄이 @@로 가는 길들이라고 설명했다. @@4의 경우 아래와 같다.

```
@\discretionary via @@2 b=1 p=50 d=12621
@\discretionary via @@3 b=291 p=50 d=103101
@@4: line 3.2- t=45102 -> @@2
```

  @@2를 거쳐서 @@4로 오는 경우의 디메리트는 12621이고 @@3을 거쳐서 @@4로 가는 경우의 디메리트는 103101이다 텍이 바보가 아니라면 어떤 길을 선택하겠는가? 당연히 @@2를 선택할 것이고, 그 증거가 바로 "@@4: line 3.2- t=45102 -> @@2"에서 "-> @@2"이다. 앞에서 설명처럼!!!

- 여기서 확실히 해둘것은 만약 문단이 @@4에서 끝난다면, 최상의 브레이크포인트의 조합은 @@1->@@2->@@4인 것이다. 그런데, @@4 다음에도 입력이 더 있다. 즉 문단이 여기서 끝나지 않고 더 있다는 것이다. 그래서 로그 파일을 자세히 살펴보니, @@2와 @@3가 경쟁관계에 있었듯이, @@4에게도 @@5라는 경쟁자가 있었다. @@4는 "document"를 "doc-"와 "ument"로 나누면서 줄바꿈을 하고, @@5는 "docu-"와 "ment"로 나누면서 줄바꿈을 한다.

- 이 부분이 하이라이트이다. 이 문단은 총 여섯개의 가능한 브레이크포인트가 있고, 마지막 여섯번째 브레이크포인트는 가능한 이라는 수식어를 띄어내야 한다. 왜냐하면, 마지막 브레이크포인트이기 때문이다. 즉 문단이 거기서 끝난다는 의미이다. @@6을 자세히 살펴보자.

```
@\par via @@4 b=0 p=-10000 d=5100
@\par via @@5 b=0 p=-10000 d=5100
@@6: line 4.2- t=24341 -> @@5
```

  마지막 줄을 살펴보면 @@6으로 가는 최상의 길은 @@5에서 부터이다. 그런데 이상하다. @@4에서 @@6으로 가는 길은 @@5에서 가는 길과 디메리트가 동일하게 5100이다 그런데 왜 @@5일까? 거슬러 올라가서 @@5와 @@4를 조사해보자.

```
@@4: line 3.2- t=45102 -> @@2
@@5: line 3.1- t=19241 -> @@3
```

비교가 되는가? @@1->@@2->@@4의 디메리트이 총합은 45102이고, @@1->@@3->@@5의 디메리트이 총합은 19241이다. 게임 끝났다. 디메리트 총합에서 게임이 안된다. 그래서 @@5가 선택된 것이다.

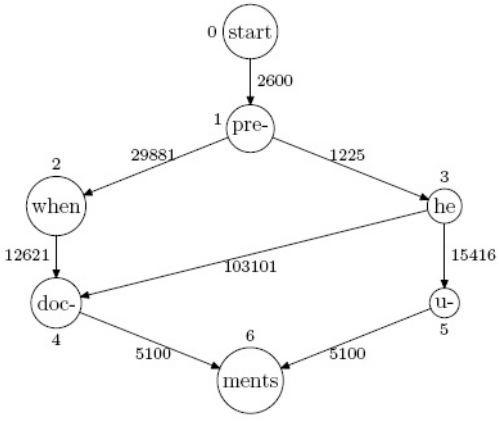
- 따라서 "'@@1 -> @@3 -> @@5 -> @@6"'이 최상의 브레이크포인트들의 조합이고 텍은 이 조합을 택해서 문단의 줄들을 만든다!!
- 정리한다. story.log 파일로 확인 해 본 결과 다음과 같다.
  - @@1 : "pre-" 에서 줄바꿈
  - @@3 : "he" 에서 줄바꿈
  - @@5 : "docu-" 에서 줄바꿈
  - @@6 : 문단의 마지막

확인 들어갑니다.

Mr. Drofnats—or "R. J.," as he preferred to be called—was happiest when he was at work typesetting beautiful documents.

어떤가? 위의 설명대로 줄바꿈이 되었는가? :))

아래의 그림은 The Advanced TeXbook에 나오는 그림으로 현재 우리가 다루고 있는 줄바꿈 알고리듬을 그래프로 나타낸 것이다. 즉, 가능한 줄바꿈지들을 노드로 그리고 노드간의 비용(cost)는 demerit로 바꾼 그래프이다. 이렇게 바꾸고 나면 줄바꿈 알고리듬은 주어진 그래프에서 @@0에서 @@6으로 가는 최단경로를 구하는 것으로 바뀌게 된다. 재미있다.



(깜빡 잊고, 연습문제가 하나 주어졌는데 그냥 지나쳤다.)

연습문제: 14.12: @@1에서 @@2로 가는 과정에서 디메리트는 29881이고, @@2에서 @@4로 가는 과정에서 디메리트는 12621이다. 디메리트 계산식에 넣어보니 틀리다. 어찌된 일인가?

> 직접계산해보자 @@1에서 @@2의 과정에서 badness는 131이고 penalty는 0이고, linepenalty는 플레인텍 기본값이 10이다. 그래서 계산하면 $(10 + 131)^2 + 0^2 = 19881$이다. 29881과

10000이 차이난다. 어디서 10000을 가져올 것인가? `\adjdemerits`를 기억하는가? :) 첫번째줄은 .2이므로 decent 하고 두번째줄은 .0 이므로 very loose이다. 따라서 이 이웃한 둘 줄은 visually incompatible하다. 전글의 설명대로 두 이웃한 줄이 visually incompatible하면 텍은 기본적으로, 구한 디메리트에 `\adjdemerits`를 더한다고 했다. 그리고 플레인텍의 `\adjdemerits`의 기본값은 10000이므로 최종 디메리트는 19881+10000=29881 이다. @@2에서 @@4로의 계산도 마찬가지이다. $(10 + 1)^2 + 50^2 + 10000 = 12621$.

story.log 파일에는 그런 경우가 없지만, 로그 파일을 살펴보다보면, b=*인 경우가 나올 때도 있다. 즉 badness가 *인 경우다. 이 뜻은 도저히 디메리트를 최소한으로 유지하면서 줄바꾸는 방법을 못찾겠다고 배째라는 뜻이다.

▶ 너무 재미있지 않습니까? 작은나무는 이 글을 쓰는 동안 너무 행복했습다. ㅎㅎㅎ – *[작은나무] [[DateTime(2006-05-16T01:32:51)]]*

▶ 세인트 앤포드*(St. Anford)* 대학교의 드로프낫츠*(Drofnats)* 교수님을 여기서도 뵙게 되는군요. *:)* 재미있게 잘 읽었습니다. – *[Karnes] [[DateTime(2006-05-16T06:01:56)]]*

▶ 드로프낫츠*(Drofnats)* 교수님, 유명하신 분인가요? 텍북에 신신찮게 나옵니다. :) 크누스 할아버지랑 친하신가보죠? 아니면, 크누스 할아버지가 만들어낸 가공의 교수님이신가요? 이 텍북이나 *taocp* 책을 읽다보면, *J. H. Quick* 이란 이름을 가진 학생이 자주 등장하는데, 크누스가 만들어낸 가공의 학생이라네요. – *[작은나무] [[DateTime(2006-05-16T06:19:21)]]*

▶

```
\def\lifo#1#2{\ifx\ofil#2 #1\ofil\fi
    \lifo{\p{#2}#1}}
\def\ofil#1\lifo#2{\fi}
\def\p#1{#1}

\leavevmode\lifo Drofnats \ofil
\bye
```

– *[Karnes] [[DateTime(2006-05-16T07:01:30)]]*

▶ ㅎㅎㅎ 그래서, "세인트 앤포드*(St. Anford)* 대학교의 드로프낫츠*(Drofnats)* 교수님" 이란 표현을 쓰셨구나! "세인트 앤포드" 만고 그냥 "스탠포드"라고 하셨어도 못 알아 챘을 겁니다. *:)* 재미있습니다. 감사합니다. – *[작은나무] [[DateTime(2006-05-16T07:12:21)]]*

11 \parfillskip

작은나무 2006-05-16T14:56:49

99–100쪽: 이전 글에서 줄바꿈 알고리듬의 결정판을 보았다. 이 14장의 목적을 거의 이룬 것이나 다름없다. 앞으로 남은 것들은 사소한 것들이다. 하지만 남아 있는 문단들에서 "당거러스 밴드" 두개짜리가 많이 보이는 이유는 무엇일까? 중요한 것과 어려운 것이 비례하는 것은 아닌 것 같다. 아니면, 남은 것들이 사소한 것이 아니던가...

줄바꿈 알고리듬에 의해서 나온 문단을 보면 폭이 모두 같고, 맨마지막만 다른데, 당연히 같거나 짧다. 이에는 특별한 트릭이 숨겨져 있는데, 아직까지 언급하지 않았다. 이와 관련해서 텍은 줄바꿈 알고리듬을 돌리기 전에 두가지 중요한 일을 한다고 한다.

1. 만일 현재 처리하고 있는 수평리스트의 마지막 아이템이 글루이면, 그 글루는 무시한다.
2. 위에서 마지막 글루를 제거하고, 그대신 세가지 아이템을 추가한다고 한다.

```
penalty10000; \hskip\parfillskip; \penalty-10000
```

이 다음부터는 원문을 그대로 소개한다. 왜냐하면, 작은나무가 대신 설명할 수도 있으나, 원문 그대로 읽어야 저자의 의도와 재치를 알 수 있기때문이다. 작은나무가 귀찮아해서가 절대 아니다. 참고로 한마디 하면, 아래 문단은 사각형이다. 즉 맨 마지막 줄도 다른 줄과 그 길이가 같다. :)

We still haven't discussed the special trick that allows the final line of a paragraph to be shorter than the others. Just before TEX begins to choose breakpoints, it does two important things: (1) If the final item of the current horizontal list is glue, that glue is discarded. (The reason is that a blank space often gets into a token list just before `\par` or just before `$$`, and this blank space should not be part of the paragraph.) (2) Three more items are put at the end of the current horizontal list: `\penalty10000` (which prohibits a line break); `\hskip\parfillskip` (which adds "finishing glue" to the paragraph); and `\penalty-10000` (which forces the final break). Plain TEX sets `\parfillskip=0pt plus1fil`, so that the last line of each paragraph will be filled with white space if necessary; but other settings of `\parfillskip` are appropriate in special applications. For example, the present paragraph ends flush with the right margin, because it was typeset with `\parfillskip=0pt`; the author didn't have to rewrite any of the text in order to make this possible, since a long paragraph generally allows so much flexibility that a line break can be forced at almost any point. You can have some fun playing with paragraphs, because the algorithm for line breaking occasionally appears to be clairvoyant. Just write paragraphs that are long enough.

익히 알고있었지만, 크누스 할아버지는 정말 재미있는 분이시다.

연습문제: 14.13: 위 설명대로라면 문단의 맨 마지막줄은 `\parfillskip`이 부족한 부분을 글루로 채운다고 했다. Ben User는 장난기가 발동해서 그 `\parfillskip`의 효과를 없애기 위해서, 문단의 맨 마지막에 `\hfilneg\par` 라고 했다. 회심의 미소를 지으며, 그러나 결과는 장난이 먹히지 않았다. 왜 일까?

> 텍은 `\par` 바로 앞에 있는 글루를 모두 무시해서 `\hfilneg`를 제거해버렸기 때문이다. :) Ben은 다음과 같이 장난을 쳤어야 했다. `\hfilneg\␣\par`.

연습문제: 14.14: \parfillskip을 어떻게 설정하면, 맨 마지막 줄의 남은 간격이 맨 첫줄의 들여쓰기 만큼 되게 할 수 있을까? 마치 $y = x$에 대한 대칭인것 처럼 보이게끔 말이다.

> \parfillskip=\parindent. 간단하다. 물론 이것이 먹히려면 만드는 문단이 충분히 길던가 아니면 짧은 문단에서도 효과를 나타낼만큼 운이 좋던가 해야한다.

연습문제: 14.15: 텍은 주어진 문단을 줄들로 나누기 위해서 일단 그 문단 전체를 입력으로 받아들여야 한다. 즉 전체 문단을 알아야 이 놈을 어떻게 줄들로 나눌지를 결정할 수 있을 것이다. 그런데 당신이 어떤 철학자나 현대 소설자들의 글을 조판한다고 해보자. 그들은 텍의 사정을 아는지 모르는지 한 문단을 200줄 이상이 넘도록 매우 길게 쓰기도 한다. 그러면 텍은 그 긴 문단을 모두 입력으로 받아들이다가 더이상 받아들일 공간이 없어서 텍시스템이 끝장날지도 모른다. 이 경우 여러분은 어떻게 해야할까? 그들에게 텍의 사정이 이러이러하니, 제발 문단을 짧게 해달라고 요청할까?

> 그 저자들이 이미 고인이 되었거나 매우 완고해서 문단을 제발 좀 짧게 해달라는 우리의 요청이 받아들여질 수 없는 상황이라고 가정하자 :) 해결책은, 대략 매 50줄마다
>
> \parfillskip=0pt\par\parskip=0pt\noindent}
>
> 를 넣는 것이다. 매우 기발한 방법이 아닐 수 없다. 이것이 동작하는 이유를 곰곰히 생각해 보기 바란다.

12 \leftskip, \rightskip

작은나무 2006-05-17T01:09:32

100쪽: 텍은 \leftskip과 \rightskip이라는 파라미터를 가지고 있는데, 이들은 이들에 정해진 글루 만큼 문단 전체를 왼쪽 혹은 오른쪽으로 옮기는 일을 한다. 플레인텍은 기본으로 이 값들을 0으로 하지만, \narrower라는 매크로를 가지고 있는데, 이 매크로는 \leftskip과 \rightskip을 현재의 \parindent 만큼 증가시키는 일을 한다. \narrower라는 매크로는 어디에 사용해야 잘 사용했다고 소문이 날까? :)

```
\narrower\smallskip\noindent
This paragraph will have narrower lines than
the surrounding paragraphs do, because it
uses the "narrower" feature of plain \TeX.
The former margins will be restored after
this group ends.\smallskip
```

위 문단은 그룹이 닫히기전에 \smallskip으로 끝나는 것에 주목할 필요가 있다. 만약에 그렇게 하지 않으면, 우리가 의도했던 \narrower의 효과는 나타나지 않을 것이다. (직접 해보시길...)

연습문제: 14.16: 한 문단 전체를 italic 이나 slanted 글꼴로 입력하면, 글자들이 오른쪽으로 약간 기울기 때문에 다른 문단들에 비해서 오른쪽으로 약간 옮겨간 느낌이 든다. 그래서 이 문단을 왼쪽으로 1pt만큼 이동해서 바로잡고자 한다. \leftskip과 \rightskip을 사용해서 해결하라.

```
{\leftskip=-1pt \rightskip=1pt <text> \par}
```

연습문제: 14.17: \centerline, \leftline, \rightline, \line 과 같은 플레인텍 매크로들은 \leftskip과 \rightskip의 영향을 받지 않는다. 영향을 받게 하려면 어떻게 하면 될까?

\line을 다음과 같이 정의한다.

```
\def\line#1{\hbox to\hsize{\hskip\leftskip#1\hskip\rightskip}}
```

이것만 정의하면 나머지 매크로들은 자동적으로 해결된다. 왜냐하면 \centerline, \leftline, \rightline이 모두 \line 매크로를 이용하여 정의되었기 때문이다.

13 \raggedright 와 줄바꿈 파라미터들

작은나무 2006-05-18T16:22:58

101쪽: \raggedright 라는 매크로를 알고 있는가? 이 매크로로 말할 것 같으면, 보통의 경우는 각 줄들의 오르쪽이 모두 맞아서 일직선 처럼 보이는데, 이 매크로를 사용하면 그렇지 않다. 즉 줄 내에서 단어들 사이의 간격이 일정하게 고정되고, 줄들의 오른쪽은 들쭉날쭉한다. 말그대로 ragged 하다. 그렇다면 이 \raggedright 매크로는 어떻게 정의되어 있을까? 한번 생각해 보기 바란다.

일단 \rightskip을 적절히 사용해서 구현했을 것이다라고 생각했다면?, 빙고! 하지만 \rightskip을 사용하는데 있어서 약간의 주의를 요한다. 예를들어 아마도 \raggedright를 "\rightskip=0pt plus1fil"로 구현했다면, 모든 줄의 오른쪽은 공백으로 채워질 것이다. 일리가 있지만, 이 방법은 그다지 좋은 방법이 못된다. 왜냐하면, 무한히 늘어날 수 있는 글루(여기서 1fil)는 매우 짧은 줄의 badness도 완전하다는 수치인 0으로 한다. (badness는 늘어날 수 있는 한계 분의 실제로 늘어난 글루 를 기반으로 하여 계산하는데, 늘어날 수 있는 한계가 무한하다면, 즉 무한대 분의 상수는 0이된다.) 따라서 이런 경우를 없애기 위해서, 즉, 짧은 줄은 보기 좋지 않다고 하기 위해서, 약간의 트릭을 쓰면 이렇다. \rightskip을 세팅하는데 어떻게 세팅하냐면, 무한대로 늘어날 수 있도록 하는 것이 아니라 그저 줄바꿈을 할 수 있을 정도로만 늘어나게 세팅하는 것입니다. 그러면, 짧은 줄이 퍼팩트하다고 하는 그런 터무니 없는 소리는 안들을테니까. :) 어떻게 세팅하면 그렇게 되냐고 묻는다면? 부록 B.를 보시라. 그리고 단어들 사이의 간격을 완전히 고정하는 것이 아니라, 약간의 유두리를 두는 것도 가능하겠다.

지금까지 살펴본 바로 텍이 줄바꿈을 하는데는 여러가지 파라미터들이 사용되는 것을 봐왔고, 그 파라미터는 오로지 줄바꿈 할때만 영향을 미친다. 예를들어, 특정 단어가 하이픈이 발생할 때 페널티를 더 물게 하기 위해서, 문단의 중간에서 \hyphenpenalty 값을 변경시키는 일 따위는 하지 말아야 겠다. \hyphenpenalty, \rightskip, \hsize등의 파라미터들은 문단의 끝에서

그 값들이 의미가 있는 것이다. 문단 중간에서가 아니라. 따라서 바꾸고 싶으면, 문단이 끝나는 곳에서 세팅하면 그 값이 반영이 되겠다. 반면에 들여쓰기의 정도를 나타내는 \parindent는 문단의 끝에서 그 값이 사용되는 것이 아니라, 현재 사용되는 수평리스트에서 그 값이 사용이 된다. 비슷하게 문단 내에서 사용되는 수식에 첨가되는 \binoppenalty나 \relpenalty등의 값들도 그 수식의 끝에서 그 값이 사용된다. 즉 문단 내에서.

14 \parshape

작은나무 2006-05-19T00:38:15

101–102쪽: 우리는 \leftskip, \rightskip을 사용하면 줄의 좌우 간격을 원하는대로 주어서 줄을 원하는 길이로 원하는 곳에 위치 시킬 수 있지만, 많이 계산해야하고 귀찮다. 즉 2% 부족하다. 이 2%를 채워주는 명령어가 바로 \parshape이다. 이름이 벌써 그런 일을 하게 생겼다. \parshape는 다음과 같이 사용한다.

```
\parshape=n i1 l1 i2 l2 ... in ln
```

먼저 n은 양의 정수로 다룰 줄의 수를 나타낸다. n 다음에 $2n$개의 <dimen>이 나온다. i1 l1 처럼 i와 l이 짝을 이루면서 나오는데, i 는 들여쓰기의 정도이고, l은 줄의 길이이다. 즉 왼쪽으로부터 i 길이만큼 떨어져서 줄을 시작하는데 그 줄의 길이는 l 이다.

연습문제: 14.18: 다음 글을 삼각형 모양으로 만들어 보라.

> I turn, in the following treatises, to various uses of those triangles whose generator is unity. But I leave out many more than I include; it is extraordinary how fertile in properties this triangle is. Everyone can try his hand.

다양한 삼각형을 만들 수 있겠지만, 여기서는 9줄의 삼각형을 만들어본다.

```
\newdimen\x \x13.4pt
\setbox1=\hbox{I}
\setbox0=\vbox{\parshape=11 -0\x0\x -1\x2\x -2\x4\x -3\x6\x
  -4\x8\x -5\x10\x -6\x12\x -7\x14\x -8\x16\x -9\x18\x -10\x20\x
  \ifdim \x>2em \rightskip=-\wd1
  \else \frenchspacing \rightskip=-\wd1 plus1pt minus1pt
    \leftskip=0pt plus 1pt minus1pt \fi
  \parfillskip=0pt \tolerance=1000 \noindent
I turn, in the following treatises, to various uses of those triangles whose
generator is unity. But I leave out many more than I include; it is extraordinary
how fertile in properties this triangle is. Everyone can try his hand.}
\centerline{\hbox to \wd1{\box0\hss}}
```

I
turn, in
the following
treatises, to various
uses of those triangles
whose generator is unity. But I
leave out many more than I include;
it is extraordinary how fertile in proper-
ties this triangle is. Everyone can try his hand.

15 Hanging paragraph와 \item

작은나무 2006-05-19T01:12:25

102–103쪽: \parshape를 이용하면 문단을 자유자재로 만들 수 있다는 것을 알았다. 하지만, 이것을 얼마나 많이 사용할 것인가? :) 아마도 그리 많지 않을 것이다. 하지만 자주 이용할 수 있는 문단 모양이 있다. 바로 \hangindent와 \hangafter 파라미터를 이용하는 것이다. 이에 대해서는 이미 3 웍에 궁은 옹인적이 있다. 그것은 그대로 적어본다.(죄송합니다.)

'\hangindent=<dimen>' 와 '\hangafter=<number>' 플레인텍에서는 굉장히 많이 사용되는 control sequence 들 입니다. x와 n을 각각 \hangindent와 \hangafter의 값이라고 하고, h를 \hsize의 값이라고 하자. 그때, $n \geq 0$이면, hanging 들여쓰기는 글 문단의 $n+1, n+2, \ldots$ 번째 줄에서 나타나고, $n < 0$이면, $1, 2, \ldots, |n|$ 에서 나타납니다. Hanging 들여쓰기란 각 줄의 폭이 일반적인 폭인 h가 아니라 $h - |x|$ 인 것을 뜻합니다; 만일 $x \geq 0$이면, 각 줄들은 왼쪽에서 들여쓰기가 되고, 그렇지 않으면 오른쪽 들여쓰기가 됩니다. 무슨 말인지 감이 오십니까? 상황이 이러하니, 참고서가 필요하겠지요?

동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세. 무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세. 무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.

다음과 같은 입력으로 위의 결과를 얻을 수 있습니다.

```
\hangindent=2cm \hangafter=-2 \noindent
동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세.
무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.
동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세.
무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.
```

(요기까지가 예전글을 그대로 복사해서 붙인 것이다. 아래부터는 아닙니다. 믿어주세요.)

Hanging paragrph가 사용되는 예는 아마도 계층적으로 항목들을 나열하는 경우에 많이 사용될 것이다. 이에 해당하는 레이텍 문법에는 \begin{itemize}...\end{itemize}이 있다. 플레인텍에도 똑같은 기능을 하는 \item 이라는 매크로가 있다. 다음과 같은 출력을 얻으려면,

1. This is the first of several cases that are being enumerated, with hanging indentation applied to entire paragraphs.
    a) This is the first subcase.
    b) And this is the second subcase. Notice that subcases have twice as much hanging indentation.
2. The second case is similar.

다음과 같이 입력하면 된다.

```
\item{1.} This is the first of several cases that are being
enumerated, with hanging indentation applied to entire paragraphs.
\itemitem{a)} This is the first subcase.
\itemitem{b)} And this is the second subcase. Notice
that subcases have twice as much hanging indentation.
\item{2.} The second case is similar.
```

연습문제: 14.19: 어느 한 \item의 항목이 하나 이상의 문단으로 구성되어 있다고 하자. 어떻게 하면 될까? (그냥 문단을 끝내면, 그 시점에서 \item의 효과도 사라지게 된다.)

> 두번째 문단부터 \item{}로 시작한다. :)

연습문제: 14.20: 위에 예로든 \item 에서 항목을 '1.'과 같은 숫자로 하는 것이 아니라 *bullet* 으로 하고 싶다.

```
\item{$\bullet$}
```

연습문제: 14.21: \item 매크로는 오른쪽 여백은 바꾸지 않는다. 왼쪽 뿐만 아니라 오른쪽도 들여쓰기를 하고 싶다.

> \hsize를 바꾸거나 \rightskip를 바꾸면 된다. 그리고 문단 끝에서 원래대로 되돌려 놓는다.
>
> ```
> \let\endgraf=\par \edef\restorehsize{\hsize=\the\hsize}
> \def\par{\endgraf \restorehsize \let\par=\endgraf}
> \advance\hsize by-\parindent
> ```

만약 \parshape 와 hanging indentation가 동시에 주어졌다면, \parshape가 우선하여 \hangindent는 무시한다. \parshape=0, \hangindent=0pt,\hangafter=1 일때, 줄의 폭이 \hsize 이면 아주 평범한 문단을 얻을 수 있다. 텍은 모든 문단의 마지막에 자동적으로 바로전의 값들을 세팅한다. 그리고 internal vertical mode 에 들어갈때도 늘 그렇다.

16 \prevgraf, \looseness

작은나무 2006-05-24T14:53:59

103–104쪽: 만일 별행수식(displayed equations)이 좀 유별나게 생긴 문단 내에 나타난다면, 텍은 그 별행수식은 정확히 문단의 세줄을 차지한다고 가정한다. 예를들어, 어떤 문단이 네 줄로 된 텍스트와 별행 수식 그리고 나머지 두줄로 되어 있다면 텍은 그 문단은 전체 4+3+2=9 즉 아홉 개의 줄을 가지고 있다고 판단한다.

텍은 내부적으로 \prevgraf라는 변수를 가지고 있고, 이 변수에 현재 텍이 처리 완료하였거나 처리 하고 있는 문단의 줄의 갯수를 기록한다. 또 당신이 텍으로 하여금 한 문단 내에서 특정한 위치에 있도록 여기게 하려고 한다면 \prevgraf에 음수가 아닌 정수를 세팅하면 된다. 위의 아홉 줄을 가진 문단을 예로 들어보자. 먼저 문단의 처음에서 \prevgraf=0이다. 별행수식을 처리하려고 하는 시점의 \prevgraf=4이고, 별행수식 처리를 끝내면 \prevgraf=7이고, 문단의 맨마지막에 도달하면 \prevgraf=9가 된다. 만일 별행 수식이 사실상 보통보다 한 줄이 더 크다면 마지막 두줄을 처리하기 전에 \prevgraf=8이라고 하면 텍은 최종 만든 줄수를 10이라고 인식할 것이다. \prevgraf의 값이 줄바꿈 알고리듬에 영향을 미칠때는 텍이 유별난 \parshape나 hangindent를 다룰 때 뿐이다.

여러분은 지금까지 줄바꿈 알고리듬을 배워오면서, 익혀야하고 알아두어야 할과 주의해야 할 것이 참으로 많다는 것을 느꼈을 것이다. 하지만 미안하게도 익혀야 할 명령어가 하나 더 있다. 바로 "looseness"라는 것이다. 앞으로 언젠가 여러분은 보다 세밀한 조정을 위해서 이 looseness가 필요할고 쓸 기회가 있다는 것을 알게될 것이다. 만일 \looseness=1이라고 한다면, 텍은 줄바꿈 알고리듬에 나온 줄의 수보다 한 줄 더 만들려고 노력할 것이다. 물론 이때, 한 줄 더 만듦으로 해서 문단의 줄들의 badness가 주어진 tolerance를 넘는 일을 없어야 겠다. 비슷하게 \looseness=2라고 하면 텍은 옵티말 줄의 갯수보다 두 줄 더 만들 것이다. \looseness=-1이면 텍은 줄을 한 줄 줄이려고 할 것이다. 지금까지의 설명을 일반화 한 아이디어는 이렇다; 텍은 먼저 앞의 글들에서 설명한 대로 주어진 줄바꿈 알고리듬대로 줄바꿈 지점들을 찾는다; 그래서 최적의 줄의 갯수 n을 구한다. 이때 만일 \looseness=l 이라면, 텍은 주어진 tolerance를 초과하지 않은 범위 내에서 가능한 $n+l$ 만큼의 줄을 만들려고 노력한다.

여러분은 간혹 \looseness=1이라고 해야할 때가 있다. 언제? 여러분이 만들고 있는 페이지가 충분히 유연한 글루(glue)를 가지고 있지 못할 때, 그 페이지에 "club line" 이나 "widow line"이 생기는 것을 피하려고 할 때에. club line 이나 widow line의 뜻은 줄에 하나의 단어만 덩그러니 있는 경우를 말한다. 문단을 만들 때, 그 문단의 맨 마지막 줄이 너무 짧아서 (즉 club line, widow line) 많은 공백이 필요한 문단의 그다지 보기에 좋은 문단이 아니다. 그래서 그러한 문단이 생기는 것을 막기 위해서 \looseness의 조정이 필요하다. 또는 문단의 맨 마지막 두 단어를 ~ 로 연결하면 하나의 단어로 끝나는 줄이 생기는 것을 방지 할 수 있을 것이다. 사실 실제 텍북에서 이 부분은 지금 설명하고 있는 내용을 실제로 적용하여 문단을 만들고 있다. 그 내용을 확인하기 위해서는 텍북 104쪽 두 번째 문단을 참고하라.

▶ 이번 모임에서 많은 분들이 특히나 작은나무가 존경하는 분들이 이 텍북읽기를 재미있게 읽어 주신다고 해서 많이 감사했습니다. 보잘 것 없는 글에 관심을 가져주시고, 글쓰는 이에게 용기를 북돋워주신 점 더욱 감사드립니다. :) – [작은나무] [[DateTime(2006-05-24T14:56:19)]]

17 Interline penalties

작은나무 2006-05-25T13:52:38

104–105쪽: 텍은 줄바꿈 알고리듬을 통하여 만들어 낸 최적의 줄들을 페이지를 만들기 위해서 현재의 vertical list에 붙여 넣는다. 이 때, 줄들 사이의 간격은 \baselineskip, \lineskip, \lineskiplimit의 값들에 의해서 결정된다. 이런 값들 뿐만 아니라 줄들 사이에 페널티를 넣기도 한다. 이 페널티를 넣음으로 해서 적절한 곳에서 쪽나눔이 되도록 한다. 줄바꿈 알고리듬과 형제 격으로 쪽나눔(page breaking) 알고리듬이 있는데, 이에 대해서는 언젠가 살펴볼 기회가 있을 것이다. :)

위에서 줄들 사이에 페널티를 넣을 수 있다고 말했는데, 이 페널티들은 어떻게 계산되는지 살펴보자: 텍이 현재 어떤 문단을 줄들로 나누었고, 그 줄의 갯수가 n이라고 하자. 그리고 j가 1보다 크거나 같고 n보다 작은 수라고 할때, j 와 j+1 번째 줄들 사이의 페널티의 값은 \interlinepenalty의 값 더하기 특별한 경우에 발생하는 페널티의 값이다. 그 특별한 경우의 페널티들을 알아보자. 먼저 \clubpenalty. 이 페널티의 값은 j가 1일 때, 즉 첫번째줄 바로 다음에 더해지는 값이다. 그리고나서 \displaywidowpenalty 또는 \widowpenalty의 값이 주어지는데 이 값은 j가 $n-1$일 때 주어지는 페널티 값이다. 즉 맨 마지막 줄에. 그리고 마지막으로 \brokenpenalty의 값은 j번째 줄이 discretionary break로 끝났을 때 더해지는 페널티 값이다. (플레인텍은 \clubpenalty=150, \widowpenalty=150, \displaywidowpenalty=50, \brokenpenalty=100으로 정한다. 그리고 \interlinepenalty의 값은 대개가 0이다. 그러나 각주(footnote)에서는 100이라는 값으로 주어지는데, 이는 그 길이가 긴 각주에서 페이지가 바뀌는 것을 방지하기 위함이다.)

위에 보기에도 이상하게 보이는 페널티들을 언급하였다. 특히 줄들 사이에 주어지는 페널티들. 페널티의 의미를 다시 한번 되새겨보자. 페널티가 무엇인가? 아주 간단하게 줄 바꿈하는데 드는 비용이다. 페널티가 크면 텍을 줄바꾸는데 주저할 것이고 반대로 작거나 음수이면 줄바꿈을 하려고 노력할 것이다. 페널티의 의미를 다시 되새기면 이 글 Interline penalties를 다시 한번 더 읽어 보자. :)

18 \vadjust, \everypar

작은나무 2006-05-26T01:17:51

105쪽: 설명의 편의를 위해서 직접 텍으로 작성한 결과를 보여줍니다. 실제 텍북에서도 저자가 그렇게 했습니다. 왜 그렇게 했는지 이해가 됩니다. 이거 말로 설명하자니 만만치가 않습니다. 백문이 불여일견!

문단 안에서 \vadjust{<내용>} 라고 하면, 텍은 '<내용>'을 현재 다루고 있는 줄 다음에 그 내용을 넣을 것이다. 언제? \vadjust라는 명령어를 포함하는 줄이 완성된 다음에. (무슨 말인지 이해가 되나요? 거참 ... 내용은 제가 써놓고도 무슨 말을 하고자 하는 것인지, 쩝.) 그러니까 이렇습니다. 제가 지금 바로 이 지점에 \vadjust{\kern5pt} 라고 입력하였습니다. 그러면 결과가 바로

지금 보고 있는 문단 처럼 된다는 말입니다. 줄간격이 조금 더 벌어져 있는게 눈에 보이죠? 더 벌어진게 바로 5pt만큼입니다. 텍이 주어진 입력에서 줄들을 만들어 가다가 어떤 줄을 만들었습니다. 줄바꿈 알고리즘에 의해서. 그런데 방금전에 만든 줄에 \vadjust가 들어있다고 합시다. 그러면 텍은 그 줄을 만든 다음에 \vadjust의 인자로 주어진 내용을 그 다음에 추가한다는 말입니다.

이제 \vadjust가 확실히 이해가 되리라 믿습니다. :)

텍에는 \every로 시작하는 재미있는 명령어들이 있습니다. 그 중에 \everypar 라는 놈이 있는데, 이 이름이 의미하듯이 매번의 \par 마다 어떤 일을 하라는 뜻의 명령어 인것을 짐작할 수 있습니다. 예를들어 \everypar={A}라고 하면, 현재 문단을 끝내고, 그 다음에 'A'를 입력하라는 뜻입니다. (좀 더 복잡한 내용이 있는데, 개념상 간단히 설명하면 그렇다는 것입니다.) 다음의 연습문제로 \everypar의 이해를 보다 확실히 하자.

연습문제: 14.29: \everypar를 이용해서 \insertbullets라는 매크로를 정의하는데, 이 매크로가 하는 일은 그룹 내의 모든 문단의 시작할 때마다, 들여쓰기 하는 곳에 bullet 를 찍는 것이다.

```
\def\insertbullets{\everypar={\llap{$\bullet$\enspace\ }
```

테스트해보자

- 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세 무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.
- 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세 무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.
- 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세 무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.

위 결과는 아래와 같은 코드로 작성된 것이다.

```
\def\insertbullets{\everypar={\llap{$\bullet$\enspace\ }
\insertbullets{%
동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.

동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.

동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.\par}
```

19 줄바꿈 알고리듬의 능력

작은나무 2006-05-26T02:56:05

"'106쪽"': 텍의 줄바꿈 알고리듬의 능력이 어느 정도인지 예를 통해서 한번 알아보자.

This is a case where the name and address fit in nicely with the review.

A. Reviewer (Ann Arbor, Mich.)

But sometimes an extra line must be added.

N. Bourbaki (Paris)

위의 결과는 아래와 같은 \signed 라는 매크로를 이용해서 얻은 결과이다.

```
This is a case where the name and address fit in nicely
with the review. \signed A. Reviewer (Ann Arbor, Mich.)
But sometimes an extra line must be added. \signed N. Bourbaki (Paris)
```

위의 글은 어떤 잡지의 리뷰란에의 포멧인데, 결과를 보면 알 수 있듯이, 현재의 줄에 리뷰하는 사람을 써 넣을 공간이 충분하다면, 그 줄에 오른쪽 정렬로 써넣고, 그렇지 않다면 다음줄에 오른쪽 정렬로 써넣는다.

\signed 매크로를 자세히 살펴보자.

```
\def\signed #1 (#2){{\unskip\nobreak\hfil\penalty50
  \hskip2em\hbox{}\nobreak\hfil\sl#1\/ \rm(#2)
  \parfillskip=0pt \finalhyphendemerits=0 \par}}
```

텍의 줄바꿈 알고리듬에 의해서, 만일 \penalty50에서 줄바꿈이 일어나면 어떻게 될까? 예전에 배울 기억을 되살려보자. 줄바꿈 지점 다음에 나오는 모든 glue는 사라진다. 따라서 \penalty50 다음의 글루 \hskip2em는 사라지고, 다음줄 처음에 \hbox{}가 나오고, 그 뒤따라 \hfil 글루가 온다. 이 의미는 줄바꿈이 일어나므로 두개의 줄이 생기는데 그 두 줄의 badness는 0이 된다; 첫번째 줄은 50의 페널티를 먹는다. 그러나 \penalty50에서 줄바꿈이 일어나지 않는다면? 리뷰와 리뷰하는 사람의 이름 중간에 2em 더하기 2fil 만큼의 글루가 생긴다. 이 역시 이 두줄의 badness는 0이 된다. 텍은 이 두가지를 모두 시도할 것이고, 시도한 결과 디메리트의 총합이 더 작은 놈을 택할 것이다. 대개는 한 줄로 끝나는 것이 더 좋다.

연습문제: 14.31: \signed 매크로에서 \hbox{}가 없다면 어떻게 될까?

> \hbox{}가 없다면, \penalty50에서 줄바꿈이 발생했을 때, 글루 \hskip2em\nobreak\hfil는 모두 사라질 것이다. 그래서 리뷰어의 이름이 왼쪽 정렬로 되어 나타날 것이다. (\hbox{}보다 \vadjust{}를 사용하는 것이 텍의 입장에서 보면 더 효율적이라고 한다.)

20 \emergencystretch

작은나무 2006-05-26T04:38:30

107쪽: 문서를 작성하다 보면 가끔 overfull 박스가 생기는 것을 경험할 수 있다. 이때, overfull 박스가 생기는 이유를 로그 파일로 살펴보고, 줄바꿈 알고리듬을 이용해서 몇가지 명령어를 추가하거나 하면 overfull 박스를 없앨 수 있다. 그런데, 이 모든 것이 귀찮으면 overfull 박스에 직통인 약, \tolerance=10000을 처방하면 그냥 해결된다. (이 대목에서 찔리는 분이 몇분 있을지도 모르겠다. :)) \tolerance=10000를 이용하면 보기에는 약간 거북하지만, 그런대로 참을 수 있는 줄들을 만들어 내기도 하지만 도저히 눈뜨고 볼 수 없는 줄도 만들어 낸다. 하지만 이처럼 무한대의 tolerance를 사용하는 것은 좋지 않은 생각이다. 왜냐하면 텍은 도처히 참을 수 없을 정도로 못생긴 줄과 그런대로 참을 수 있는 줄을 구별할 수 없기 때문이다. \tolerance=10000라고 하면, 텍은 줄들을 만들어 낼때, 모든 badness를 한 곳으로 몬다. 여러 줄들에 안좋은 badness가 적절히 분배되어 평균적으로 보기에 안좋은 줄들을 만들어 내는 것이 아니라, 꾹 참고 줄들을 보기 좋은 모양으로 만들어 내다가 그동안 참았던 badness를 모두 하나의 줄에 쏟아 붇는 것이다.

즉 분노가 폭발하는 것이다. :) 왜냐하면, 그렇게 하는 것이 줄바꿈 알고리듬의 입장에서 볼 때, 디메리트의 총합이 최소가 되기때문이다. 다행스럽게도 텍은 `\emergencystretch`라는 값을 가지고 있다. 이 값은 각 줄들의 badness와 디메리트를 계산할 때, 각 줄들에 더해지는 값이다. 이 값을 더하지 않으면 overfull 박스가 생기는 것을 피할 수 없는 경우에 이 값을 사용함으로 그것을 피할 수 있는것이다. 만약 `\emergencystretch`의 값이 양수라면, 줄바꿈 알고리듬을 끝내기 전에 해당 문단을 한번 더 살펴본다(scan). 언제 그런고 하니, 줄바꿈 알고리듬의 첫번째 패스에서, `\pretolerance`나 `\tolerance`를 만족하면서 줄바꿈을 하는 지점들을 못찾았을 때 말이다. `\emergencystretch`의 값을 충분히 크게하면 주어진 `\tolerance`는 절대로 초과하지 않는다고 장담해도 좋다. 따라서 줄바꿈 알고리듬으로 절대로 불가능한 경우를 제외하고는 overfull 박스를 피할 수 있다.

Afterwords

이것으로 5월의 목표였던 줄바꿈 알고리듬을 마칠까 합니다. 이 글은 *TEXbook*의 제14장 “How TeX Breaks Paragraphs into Lines”를 읽어나가면서 작성한 글입니다. 최대한 번역으로 흘러가는 것을 막으려고 노력했지만, 일부 글들에서는 번역 투의 글을 많이 발견할 수 있을 것입니다. 제 글쓰기 실력이 괜찮다면 피할 수도 있었겠지만..., 글쓰기는 둘째치고, 제대로 이해를 못해서 번역 투의 글이 나온 것이 있을 수도 있습니다.

5월의 남은 기간은 제가 쓴 글들을 다시 읽어보며, 수정을 하고자 합니다. 아마도 말도 안되는 문장도 많을 것입니다. `:)`

읽어주셔서 감사합니다.