Miscellanies

西二闲画生
Department of Life,
Zhonghua Uni.,
WH, China
zhu@riseup.net

February 20, 2019

Preface

LaTeX typesets text as *What You See Is What You Type* compared with *What You See Is What You Get*. It gives finer control over text layout with a host of commands and add-ons. This book firstly demonstrates how LaTeX supports Chinese typesetting and then moves on to syntax. A bit of ACTeX with Emacs configuration and key bindings are dicussed as well. Gradually, topics from different disciplines (i.e. math and language) are introduced in separate parts.

Priorly, I just planned to learn LaTeX through trial and error. As it progressed, I found LaTeX an awesome typesetting tool, creating beautiful text output. Hence, I decided to make it a book containing different parts, covering formal subjects like math, physics, etc. Ultimately, I want to maintain a centralized offline jottings which can be testified by the book title *Miscellanies*.

Contents

Part I

LaTeX

Chapter 1

LaTeX Resources

1.1 Find References

In part 2 on page 5, we have a simple clarification on how to insert Chinese characters in English articles. We will focus on LaTeX itself in this part.

Please find more appropriate at LaTeX ctan. Also, please find anwsers at [1]. You'd better go through the tutorial series [2]. My personal webiste [3] might give some ideas.

1.2 AUCTeX

1.2.1 Inserting

Have a look at LaTeX

1.2.2 LaTeX Sources Formating

Use `C-c C-q C-s` to format LaTeX *section* code. `C-c C-q C-p` is to format *paragraph* text.

1.2.3 Font Specifiers

Use `C-c C-f C-i` to insert *italic* text. Smililary, Use `C-c C-f C-b` to insert *bold* text. Read more at Insert Font Spcecifiers.

Binding	Function
C-c C-e	insert environment
C-c C-s	insert section
C-c C-m	macro (command)
M-return	another item
C-c C-c	compile LaTeX
C-c C-v	preview output
C-c =	show document layout by RefTeX
C-c C-o C-b	fold the buffer
C-c C-o b	unfold the buffer
C-c C-r	compile selected region
C-c C-b	compile current buffer

Table 1.1: AUCTeX Bindings

C-c C-f C-i	italic
C-c C-f C-b	bold
C-c C-f C-e	emphasize
C-c C-f C-s	slight italic
C-c C-f C-r	roman
C-c C-f C-f	sans
C-c C-f C-d	delete font formating

Chapter 2

汉语

对字体的引用要注意，如果一个字体族 family 有多个变种[1]，请引用字体的 postscriptname.

默认情况下，LaTeX 会把文档里用到的字体子集内嵌到生成的 PDF 里，可以通过命令行 *pdffonts filename.pdf* 查询，结果如 A 所示。

此例说明，对应文档内嵌了七个字体子集。LaTeX 文档通常只会用到字体文件的部分 code point，所以只需要内嵌使用到的部分即可。每个字体的前面都有一串无规则字母，代表对应字体文件被内嵌的子集。还有一列叫 sub 也说明是否是子集。

2.1 CJK 包

在英文文档里插入少量中文

2.1.1 CJK 和 CJK* 环境

在英文文档里用 CJK 包的 CJK 或 CJK* 环境插入少量汉字。后者会忽略汉字后面的空格，推荐使用。

2.1.2 字体

bsmi 字体可能没有，尝试换一个如 gkai, gbsn 等。

[1] 如 regular, bold, italic 等。

2.2 xeCJK 更好

xeCJK 在处理细节上更好，关键是可方便设置中英文字体。用 xeCJK 则不需要特殊环境包裹汉字，只需在 “导言区” 设置好中英文字体即可。

xeCJK 的 indentfirst 选项：

```
\usepackage[indentfirst]{xeCJK}
```

已过时，推荐直接用 indentfirst 宏包：

```
\usepackage{indentfirst}
```

2.3 fontspec 和 ctex

还有 *fontspec* 和 *ctex* 包可以实现中文输入，更多详情请看 LaTeX post.

- `fontspec` sets western or Chinese fonts (i.e. main, sans, serif, bold, italic, and bolditalic).
- `CJK` sets Chinese, Japanese, and Korean fonts, punctuation etc. for TeX engine.
- `xeCJK` similar tof `CJK` but for XeTeX engine.
- `CTeX` sets `CJK`/`xeCJK` document layout.

2.4 中英文空格

我们想要的效果是英文字符前后自动加上空格。xeCJK 则会自动处理，但是效果不是非常好。

CJK 会忽略 TeX 源码里的空格。用 CJK 宏集里的 CJKspace 包，源码里英文后空格会保留，中文后空格依然被 CJK* 压缩。还有一个方法是在需要空格的地方用 tilde ∼.

2.5 行首缩进

中文习惯每段行首缩进两个汉字。英文有两种缩进格式。默认是每小节的第一段缩进，后面的段不缩进，段间没有空行。另一种是不缩进，每段间留有空行。

本文档只插入少量中文作例子，固保留英文默认缩进方式。如要换成中文缩进，则用 indentfirst 包，使每节第一段也缩进，如下：

```
\usepackage{indentfirst}
\setlength{\parindent}{2em}
```

如若换成英文第二种缩进，则用parskip 包，skip 表示相临两段之间的间隙大小：

```
\usepackage[parfill]{parskip}
```

也可用相对应的命令实现：

```
\setlength{\parindent}{0pt}
\setlength{\parskip}{1ex plus 0.5ex minus 0.2ex}
```

2.6 局部字体命令

前面提到的方法都是全局生效，有没有类似 `\emph{}` 和 `\textit{}` 这种监时改中文字体的方法呢？

在 *xecjk.tex* 中有如下代码：

```
% set new font family
\setCJKfamilyfont{zhyahei}{msyh}[
  Path = fonts/,
  Extension = .ttf,
  BoldFont = {*bd}
]

% create font family command alias
\NewDocumentCommand{\yahei}{}{\CJKfamily{zhyahei}}
```

Listing 2.1: New font family

根据需求定义一个新的字体族 family 名 *zhyahei*，新字体族的引用方法是：

```
{\CJKfamily{zhyahei}{这里输中文}}
% -or-
{\CJKfamily{zhyahei}这里输中文}
```

Listing 2.2: Inline Chinese fonts

为了方便，我们还定义一个新 NewDocumentCommand 命令，便于快速引用。`{\yahei{这是新定义的雅黑字体样例。}}` 的效果如：**这是新定义的雅黑字体样例。**

特别注总，命令要放在一个 group 里，即用 `{}` 围起来，否则从当前位置起，所有中文字体都改了。实际，fontspec 提供类似功能，具体请参考 fontspec 文档。如：

`\fontspec{⟨font name⟩}[⟨fontfeatures⟩]` 和 `\newfontfamily`

Chapter 3

LaTeX Tutorial

3.1 TeX Units

`mm`	millimetre $\approx 1/25$ inch	␣
`cm`	centimetre $= 10$ mm	␣␣
`in`	inch $= 25.4$ mm	␣␣␣
`pt`	point $\approx 1/72$ inch $\approx \frac{1}{3}$ mm	‖
`em`	approx width of an 'M' in the current font	␣
`ex`	approx height of an 'x' in the current font	␣

Figure 3.1: TeX Units

3.2 Special characters

The following 10 characters has special meanings in LaTeX.

```
& % $ # _ { } ~ ^ \
```

Outside *verb*, the first 7 should escaped by *backslash*. Of the last three, we use LaTeX macros, namely `\textasciitilde`, `\textasciicircum` and `\textbackslash`.

```
1 \& \% \$ \# \_ \{ \} \textasciitilde{} \textasciicircum{} \
      textbackslash{}
```

Listing 3.1: Special Characters

Specially, LaTeX uses tilde ~ as a non-breaking space. You would usually use non-breaking spaces for punctuation marks in some languages, for units and currencies, for initials, etc.

3.3 verbatim and verb

When to show raw text without LaTeX command being executed, use *verbatim* environment or *verb* command. Both have an asterisk alternative like `\begin{verbatim*}` and `\verb*`. With extra *, LaTeX will typewrite space as ␣.

You can't use `\verb` in other command's parameters. For example, this is not allowed:

```
\href{https://tex.stackexchange.com/a/23653}{When should we use
  \verb|\begin{center}| instead of \verb|\centering|?}
```

Listing 3.2: Illegal verb

Instead, we use `\texttt` and escape special characters [1] like:

```
\href{https://tex.stackexchange.com/a/23653}{When should we use
\texttt{\textbackslash{}begin\{center\}} instead of \texttt{\
    textbackslash{}centering}?}
```

Listing 3.3: texttt

3.4 item list

Using lists is quite straightforward and does not require you to add any additional packages. For unordered list, LaTeX provides the *itemize* environment and for ordered list there is the *emunerate* environment. With both environments, elements should be declared beginning with `\item` command. We can use `M-Enter` binding to insert this command automatically.

Take a look at unordered list3.4:

```
\begin{itemize}
\item One
\item HKUST
\item HUST
\end{itemize}
```

- One
- HUST
- HKUST

Here is an ordered list 3.4:

[1] For instance, backslash and {.

1. Jim
2. Gray
    - This is a
    - nested unordered
    - list within ordered list
3. Who are you?

```
\begin{enumerate}
\item Jim
\item Gray
  \begin{itemize}
  \item This is a
  \item nested unordered
  \item list within ordered
    list
  \end{itemize}
\item Who are you?
\end{enumerate}
```

3.5 Code listings

Although *verb* and *verbatim* enclose raw text, they don't highlight source code like C, Java etc. This is where *listings* package come into usage.

To insert a code block, use *lstlisting* environment like

```
\begin{lstlisting}[language=bash,caption={Bash},frame=single]
```

Here is an example:

```
#!/bin/bash
echo 'Hello, world!'
```

Listing 3.4: Bash

Furthermore,

```
\lstinputlisting[language=C,frame=tb,basicstyle=\scriptsize\ttfamily]{
    helloworld.c}
```

Listing 3.5: Include code file

command includes code source file:

```
# include<stdio.h>

int main(void *) {
  printf("Hello, world!\n");
  return 0;
}
```

Listing 3.6: C98

This is useful when the code needs updating frequently.

To prevent *lstlisting* from breaking pages, we should wrap it within *minipage* environment as 3.9.

Sometimes, code line is too long to fit page width, leaving tailing part out of page. We could modify *basicstyle* to one of *footnotesize*, *scriptsize* and *tiny*. Check long text line example A. For more options, read Font Size and font size and point (pt) relation.

This is LARGE text while this is Huge.

3.6 Straight Quotes

By default, *listings* typesets both single and double as *back curve ones*.

We should import *textcomp* package and *fontenc* package with T1 option. Then, enable *upquote=true* for code block. Read more at How can I get straight double quotes in listings?

This method, however, is discouraged as *fontenc* would override the whole document LaTeX font settings. The underlying cause is *serif (roman)* font. Default roman fonts does not provide straight quotes. We should tell *listings* to use *monospace* font instead by

```
\lstset{basicstyle=\ttfamily}
```

Read more at Consolas: Straight Quotes.

3.7 floating

3.7.1 Figure

First, we import *graphicx* package and then set *graphicspath* in preamble part:

```
\usepackage{graphicx}
\graphicspath{{figs/}{./}}
```

Listing 3.7: graphicx

graphicx accepts EPS, PDF, PNG or JPEG formats. *graphicspath* defines a list of directories relative to that of *master.tex*. By this means, we can access images anywhere without restriction like `\graphicspath{{../../img/}}`. The trailing slash cannot be omitted.

Then, we use command *includegraphics* like

```
\includegraphics[width=.5\textwidth,scale=0.1]{myimg}
```

Notice that we do not write figure file extension. Here is the output:

With bare *includegraphics*, if remaining space within current page does not hold the figure, LaTeX would start a new page instead, leaving current page partially emtpy. The method is to wrap it with *figure* environment:

```
\begin{figure}[tbp]
  \centering
  \includegraphics[width=\textwidth]{Boobs}
  \caption{Boobs}
  \label{fig:Boobs}
\end{figure}
```

Listing 3.8: Figure Floating

We call this technique as *floating*. Apart from *placement specifier* benefits, we can define *label* and *caption* within *floating* environment. Note that `\label{}` command must come **after** `\caption{}` command since the reference number is generated by `\caption{}`. For details, refer to Why does an environment's label have to appear after the caption?.

The `tbp` [2] specifies position of floating environment.

Spec	Permission to place the float ...
`h`	*here* at the very place in the text where it occurred. This is useful mainly for small floats.
`t`	at the *top* of a page
`b`	at the *bottom* of a page
`p`	on a special *page* containing only floats.
`!`	without considering most of the internal parameters[a], which could otherwise stop this float from being placed.

Figure 3.2: Floating Placement Specifiers

Here is a nice boobs example 3.3:

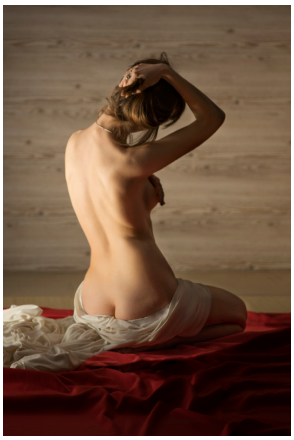
Figure 3.3: boobs

Sometimes, it's necessary to put figures side by side for clear comparasion. We use *subfigure* within *figure* like 3.4. Please be noted that we add non-breakable symbol ~ between *subfigure* to leave some space between figures.

3.7.2 Table

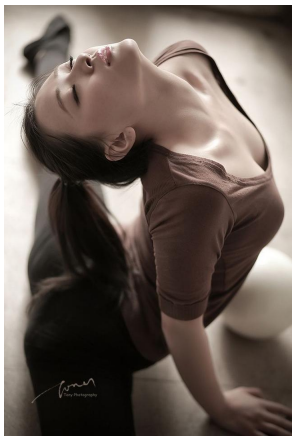
Similarly, we can wrap *tabular* within *table* environment to *float* table 1.1 like 3.9.

[2] *tbp* is the default *placement specifier*. We could set it to *!htbp*.



(a) Girl A

(b) Girl B



(c) Girl C

Figure 3.4: Three Girls

```
\begin{table}[!htbp]
  \centering
  \begin{tabular}{|l|r|}
    \hline{}
    Binding & Function \\ \hline \hline{}
    \verb|C-c C-e| & insert environment \\ \hline{}
    \verb|C-c C-s| & insert section \\ \hline{}
    \verb|C-c C-m| & macro (command) \\ \hline{}
    \verb|M-return| & another item \\ \hline{}
    \verb|C-c C-c| & compile \LaTeX{} \\ \hline{}
    \verb|C-c C-v| & preview output \\ \hline{}
    \verb|C-c =| & show document layout by RefTeX \\ \hline{}
    \verb|C-c C-o C-b| & fold the buffer \\ \hline{}
    \verb|C-c C-o b| & unfold the buffer \\ \hline
  \end{tabular}
  \caption{AUCTeX Bindings}
  \label{tab:auctex-bindings}
\end{table}
```

Listing 3.9: Table Floating

In *tabular* environment, we align column texts as ***l**eft*, ***c**enter* and ***r**ight*. What about decimal number cell? Idealy, we want to align numbers at dot, where package *siunitx* plays a role by **S** 3.1 alignment specifier.

```
\usepackage[round-mode=places,round-precision=3]{siunitx}
```

Listing 3.10: Decimal Alignment

To make prettier table, we use package *booktabs* which provides *toprule*, *midrule* and *bottomrule* to override [3] dull *hline*.

[3] It is not replace!

List 1 α	**Decimal 2** β	**Letter 3** γ
1	1385.100	a
2	3.067	b
3	87.369	c

Table 3.1: Number Alignment at Dot

$Ampere$ A	$Voltage$ V	$Energy$ J
3	5	45
5.303	4.871	136.955

Table 3.2: Table automation from .csv file.

To make more complex tables, we can use package *multirow* and *multirow* and *multicolumn* environment respectivelly.

When typesetting large tables, it is tedious and error-prone which could be avoided by *pgfplotstable* package that generate tables from external *.csv* file. Programs such as Excel, OpenOffice Calc or even emacs org-mode can export data sheets as *.csv* files. The *pgfplotstable template* A.3 generate table 3.2.

3.7.3 Plot

The *pgfplots* package is a powerful tool, based on *tikz* package, dedicated to create scientific graphs. Similar to *pgfplotstable*, *pgfplots* can read date from *.csv* file.

Since *pgfplots* is based on *tikz*, the *axis* environment should be enclosed within *tikzpicture* environment. Template A.3 generates plot 3.5.

addplot plots 2D figure, for 3D, we use *addplot3* 3.6.

Besides *pgfplotstable* and *pgfplots*, we could aslo use *tgiz* package or *circuitkiz* to `\draw` pictures from within your LaTeX document.

3.8 center and centering

The main difference between *center* environment and *centering* command is the former leave vertical space before and after it while the later would not.

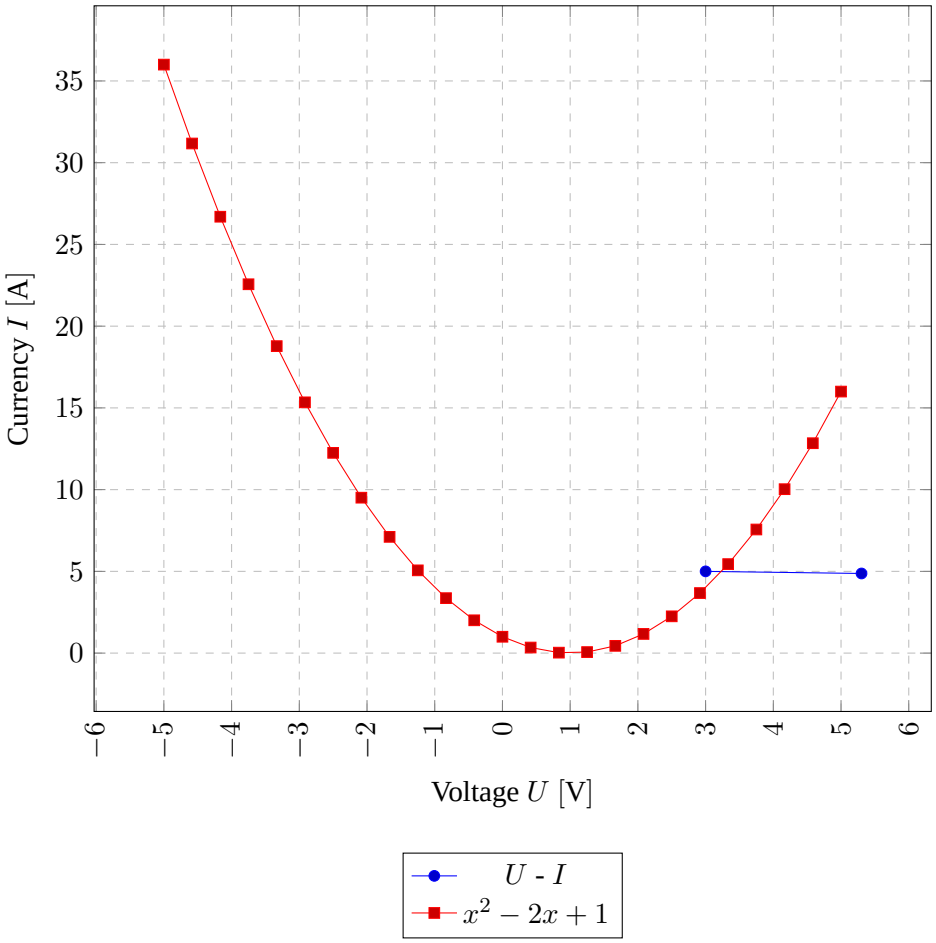


Figure 3.5: pgfplots by table csv file

We usually use `\centering{}` command within a group by curly braces, figure, table, or `\begingroup \centering ... \endgroup` 3.11.

```
{\centering{centering require line break by \par, emtpy line or \\
before closing the group, otherwise text following the group wound
be centered either.}\\
}
```

Listing 3.11: centering within braces

centering require line break by `\par`, emtpy line or `\\` before closing the group, otherwise text following the group wound be centered either.

This line is not centered! Check When should we use `\begin{center}` instead of `\centering`? and center vs. centering

Exmple using the mesh parameter

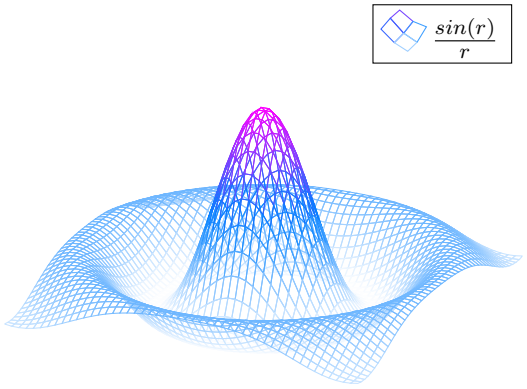


Figure 3.6: pgfplots 3D

3.9 Math Equations

To insert simple inline equation, we wrap it by $ like `$c^2=a^2+b^2$`: $c^2 = a^2 + b^2$.

For tall or deep inline math expressions or sub expressions, we can enclose them by `\smash` command. This makes LaTeX ignore the height of these expressions. This keeps the line spacing even like: $d_{e_{e_p}}$ followed by $h^{i^{g^h}}$.

This is an example without `\smash`. You will find line space is much bigger. A $d_{e_{e_p}}$ expression followed by a $h^{i^{g^h}}$ one.

If we want equaton occupy a whole line, then wrap it by double $$ or single bracket `\[\]`. The equation will be centered automatically.

```
$$(1+x)^n=\sum_{k=0}^n\binom{n}{k}x^k$$
```

Listing 3.12: Equation in new line

This is the output:

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

Let's have another example:

$$A_1 + A_{100}$$

After examing the example above, you are recommended to tune AUCTeX 1.2 configuration a little bit. Enable math mode manually by `C-c ~` or globally into Emacs startup:

```
(add-hook 'LaTeX-mode-hook 'LaTeX-math-mode)
```

Listing 3.13: LaTeX Math Mode

We just prepend ` and type your desired math symbol. AUCTeX automatically completes it. If given a prefix argument C-u, the symbol will be surrounded by dollar signs. For example, if you type C-u ` b, AUCTex will typeset β. Read AUCTex manual Entering Mathematics for more details.

The $ method by TeX does provide some basic mathematics features, but it is limited. we should \usepackage{amsmath}. What is more, we can \usepackage{amssymb} to make math symbols look shiny.

A bit further, we usually want to label and number a equation so that we can refer to it somewhere else. Futhermore, it is better for a long equation to occupy a new line. To do this, enclose equation with *equation* environment like (3.1). Einstein says

$$E = mc^2 \tag{3.1}$$

In order that multiple equations align properly at *ampersand &*, we use *align* or *align** instead. (3.9). Each single equation must be separated by *linebreak*

. The version with asterisk just removes equation numbers.

$$E = mc^2 \tag{3.2}$$
$$F = ma \tag{3.3}$$

We use *aligned* environment to align induction or substitution lines of a single equation. Also, *aligned* should be enclosed within an equation environment.

$$\begin{aligned} f(x) &= x^2 \\ \frac{1}{x} &= g(x) \\ F(x) &= \int_b^a \frac{1}{\sqrt[3]{x}} \end{aligned}$$

Compared to *aligned*, *align* and *align** introduce extra space between lines to make the output cleaner. Therefore, irrespective of single equation or multiple equations, we'd better use *align* and/or*align**.

matrix environment must be enclosed by equation marker & or *equatioin* environment.

```
$
\begin{matrix}
  1 & 0 \\
  0 & 1
\end{matrix}
$
```

Listing 3.14: Matrix

Similary, matrix breaks line by \\ and & separates columns. Here is a *bmatrix* example:

$$A_{m,n} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \tag{3.4}$$

Recall that $$ or \[\] let an equation placed on a new line without label. They are equivalent to the `\begin{equation*}` environment.

3.10 Document Structure

3.11 Boxes

Everything in LaTeX is embedded in boxes, from letter to words, *tabular* to *includegraphics*. TeX builds pages by gluing [4] boxes together according to the default TeX rules, default LaTeX rules, or document commands.

Box is of paramount importance to get the base of LaTeX typestting. Boxes are placed relative to other boxes, while visible elements are placed relative to the boxes which contain them.

Let's have a look at letter box illustration 3.7.

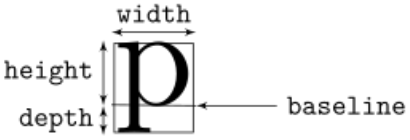


Figure 3.7: Letter Box

[4] Squeeze and/or Stretch

`\parbox` is a box to wrap text into lines and lines are broken into pages.

```
\parbox[pos][height][contentpos]{width}{text}
```

Listing 3.15: parbox box

width is a forced parameter to define box width[5] This argument can be relative value like `.8\textwidth`, absolute value like `5ex`, or LaTeX and TeX macros like `\width`. Similarly, we set *height* in the same way.

pos selects which *baseline* of text to align with neighbouring box. It can be **c**enter, **t**op, and **b**ottom. For details, read the link above. Check 3.8 for real effect.

This is a text line before *parbox* This a text within *parbox*. Please check text box and text alignment carefully. This a tex line after *parbox*.

contentpos positions the contents of the box within the box which is straightforward. It only takes effect when box is larger than texts it encases.

However, if the *contentpos* is present and not the same as *pos* and *pos* is not **c**enter, the `\parbox` will align at its borders instead of text baseline 3.9.

This is a text line before *parbox* This a tex line

This a text within *parbox*. Please check text box and text alignment carefully.

after *parbox*.

We also have *minipage* environment box which is almost identical to `\parbox`. The difference between a *minipage* and a `\parbox` is that you cannot use all commands and environments inside a `\parbox`, while almost anything is possible in a *minipage*. Hence, without special requirement, we use the later one.

```
1 \begin{minipage}[pos][height][contentpos]{width} text \end{minipage}
```

Listing 3.16: minipage box

Additionally, there are *mbox*, *makebox* and *framebox*.

In order to let multiple boxes stand side by side, we should make sure their *width* in total is less than 1, like 3.4.

[5] It is **not** text width.

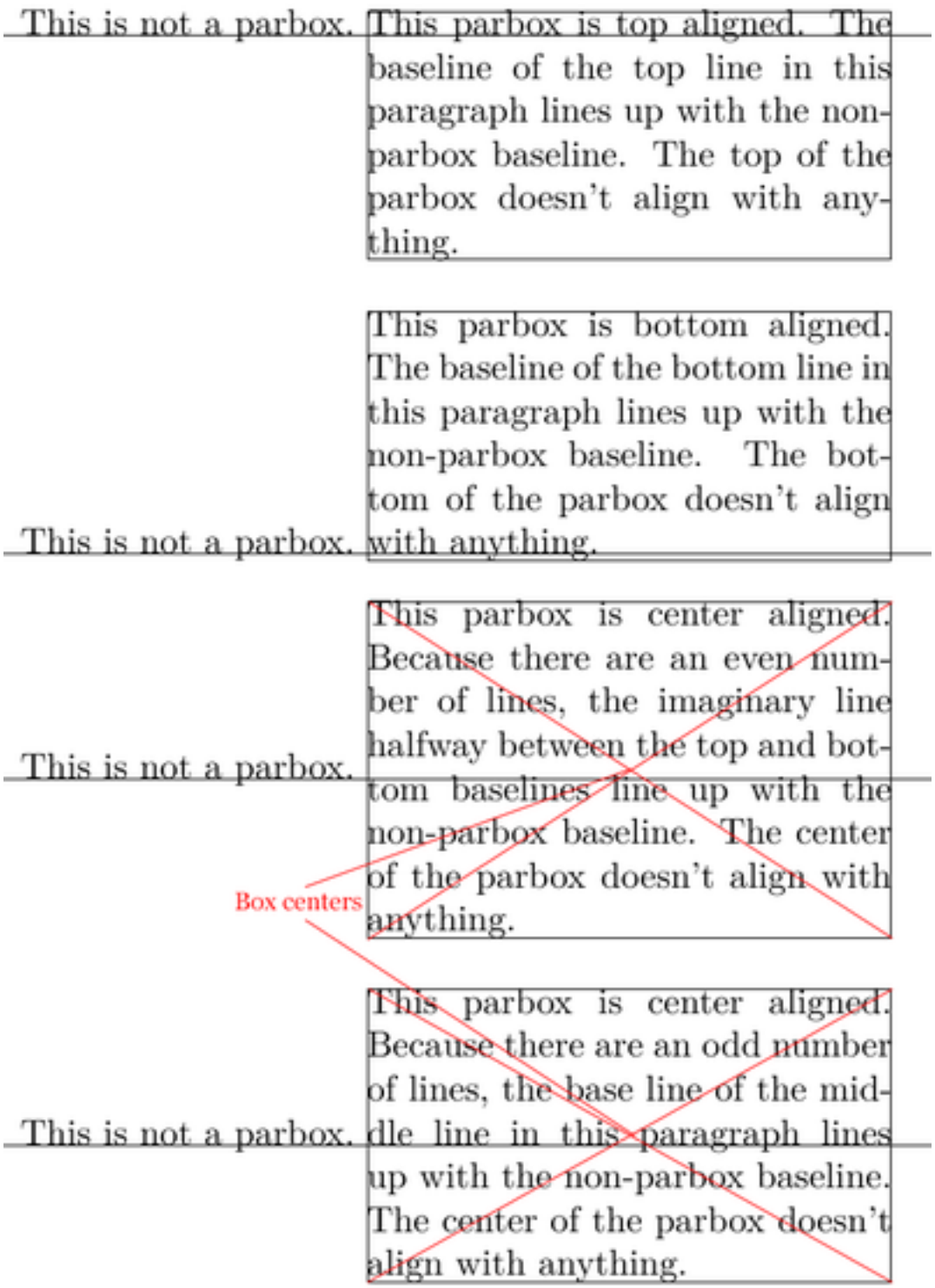


Figure 3.8: parbox baseline alignment

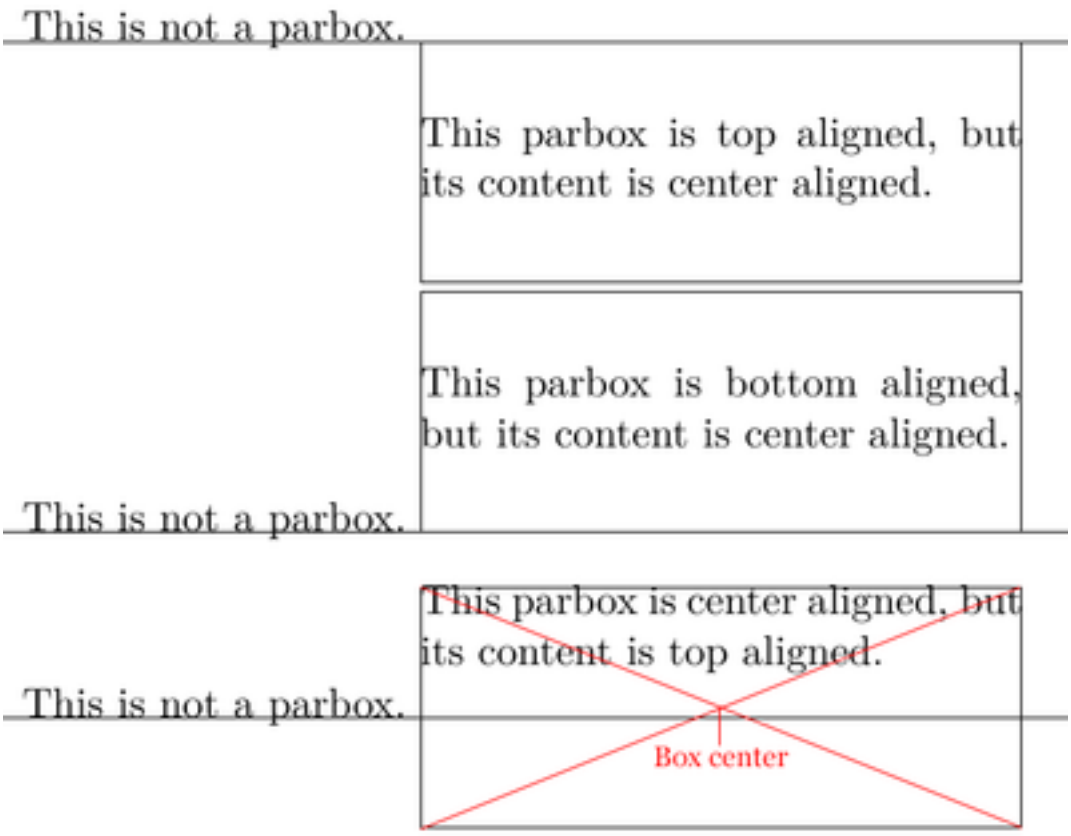


Figure 3.9: parbox border alignment

3.11.1 input and include

Like other programming language, LaTeX supports splitting large document into different parts and importing them individually.

Use `\include{filename}` in the *document body* to insert the contents of another file named *filename.tex*. Note that LaTeX will start a new page before processing the material input from *filename.tex*. *include* is commonly used for *chapter* section.

include's sibling command `\includeonly{filename1,filename2,...}` is used in *preamble* part to insert only a subset of *included* files.

Alternatively, `\input{filename}` is allowed in both *preamble* and *body* parts. It does not start a new page like *include* but allow recursive *include* which is impossible for *include*.

No matter which you choose, please omit LaTeX file extension *.tex*.

3.11.2 biblatex biber

Generally speaking, *biblatex* and *natbib* are packages that handle LaTeX citations in *.tex* source file. However, reference entries are stored separately in external *.bib* file. Hence, we require an intermediate tool to format *.bib* into *.tex* source. That's when we meet *biber* and *bibtex* which are named as *biblatex backend*.

Firstly, we should use *biblatex* package: `\usepackage[backend=biber]{biblatex}` and point out the external *.bib* file to import: `\addbibresource{myref.bib}` in preamble part. Do not omit *.bib* extension.

Secondly, in the end of LaTeX source (but before `\end{document}`), we print all reference entries: `\printbibliography[heading=bibintotoc]`.

Then, cite as you go. Before that, we should have a look at *.tex* format. The bibliography files must have the standard *bibtex* syntax. Here is a reference entry:

```
@article{einstein,
    author = "Albert Einstein",
    title = "{Zur Elektrodynamik bewegter K{\"o}rper}. ({German})
    [{On} the electrodynamics of moving bodies]",
    journal = "Annalen der Physik",
    volume = "322",
    number = "10",
    pages = "891--921",
    year = "1905",
    DOI = "http://dx.doi.org/10.1002/andp.19053221004",
    keywords = "physics"
}
```

Listing 3.17: BibTeX entry sample

`@article` tells the reference is an article. We also have `@book`, `online` etc. *einstein* is the reference label that we will *cite* with: `\cite{einstein}`.

3.11.3 RefTeX

RefTeX has been bundled and pre-installed with Emacs since version 20.2. We just add to Emacs:

```
; with AUCTeX LaTeX mode
(add-hook 'LaTeX-mode-hook 'turn-on-reftex)
; with Emacs latex mode
(add-hook 'latex-mode-hook 'turn-on-reftex)
```

Listing 3.18: Enable RefTeX

To make cross-reference clickable (i.e. table of contents), use package *hyperref*.

Although `C-c C-m` can prompt for `\cite` command, we'd better use RefTeX instead. RefTeX wraps itself round four LaTeX macros: `\label`, `\ref`, `\cite`, and `\index`, making the process more intelligent.

As mentioned earlier, RefTeX binding `C-c =` displays document layout in a newly created buffer.

To *cite*, we use `C-c [` binding and press `^M` [6]. Afterwards, type a *regular expression* to search reference entries in *.bib* file. Please be noted, there is no completion prompt for *regular expression*.

For reference to objects (i.e. figures) within the document, we use `C-c )`. For details, please visit RefTeX in a Nutshell. RefTeX seems not to support *href* to URLs.

Sometimes, RefTeX cannot find newly created labels, references etc. We should tell it to *reftex-parse-all* or *reftex-parse-one* to parse LaTeX source files.

[6] It is Enter key or Ctrl and letter 'm', not Emacs Meta Alt key.

Part II

Mathematics

Chapter 4

Combinatorics

Combinatorics, namely *Combinatorial Mathematics*, mainly studies *permutation* and *combination* of a *set* or *multiset*.

组合数学里经常要用到**映射**的概念，在后文的描述里，会多次出现**对应**这样的字眼，表示两个集合间元素的映射。通常为了方便分析，把元素、对象的不同用**编号**、**颜色**等标签来表示。

4.1 Set and Multiset

组合数学问题总要对某个集合的元素进行操作，虽然我们通常不需要特别说明。如 10 个人进行全排列，那么对应的集合由这 10 个人组成。

普通集合如：

$$S = \{ a_1, a_2, a_3, \ldots, a_n \},\ i \in \mathbb{Z} : i \in [1, n]$$

表示有 n 种**不同**元素，并且每种元素只有一个，称 S 为**单重集（合）**。

更一般的，**多重集（合）**：

$$M = \{ n_1 \cdot a_1,\ n_2 \cdot a_2,\ \ldots,\ n_k \cdot a_k \}$$

突出元素**种类**，即 a_i 表示第 i 种元素，而 n_i 则表示第 i 种元素的个数。通常元素总个数用 n 表示：

$$n = n_1 + n_2 + \ldots + n_k$$

不难发现，单重集是多重集的特例，此时

$$k = n,\ n_i = 1,\ i \in \mathbb{Z} : i \in [1, k]$$

如果多重集里每种元素有无限个，则记为:

$$M = \{ \infty \cdot a_i,\ \infty \cdot a_2,\ \ldots,\ \infty \cdot a_k \}$$

普通集合和普通排列组合对应，多重集合和多重排列组合对应，这也是在正式介绍排列组合前先说下集合的定义。

排列组合里，集合元素通常都具体化为**不同**颜色小球，组合表示从集合里取小球出来，而排列表示进一步把取出的小球排队，或放进**不同**的盒子里，此时盒子的编号代表队列从头至尾的位置号。

注意符号表示的不同。单重集里 n 既表示元素种类和个数。而多重集用 k 表示元素种数，用 n 和 n_i 表示个数。注意符号表示的习惯:

$$a_i,\ n_i,\ k,\ n,\ S,\ M$$

4.2 Counting Principles

加法原理:设集合 S 可以划分成若干不相交的子集 $S_1,\ S_2,\ \cdots,\ S_m$,则:

$$|S| = |S_1| + |S_2| + \cdots + |S_m|$$

乘法原理: 设集合 S 是序偶 (a, b) 的集合，其中 a 来自于集合 A, b 来自于集合 B. A 中每个元素 a 都要与 B 中每个元素 b 配对。则:

$$|S| = |A| \times |B|$$

减法原理: 设集合 A 是全集 U 的子集，补集[1] $\overline{A} = U \setminus A = \{ x \,|\, x \in U,\ x \notin A \}$, 则:

$$|A| = |U| - |\bar{A}|$$

除法原理: 设 S 是有限集，被划分为 k 个两两不相交的部分，每部分皆有 m 个元素，则:

$$k = \frac{|S|}{m}$$

后面会发现，很多应用都以单重集的排列、组合为原子操作，再结合计数四原则完成。

[1]用 overline, bar, complement, 或 smallsetminus 命令表示。后者的好处是同时指明了全集和子集。

4.3 Permutation of Sets

定义：从 n 个元素的单重集合 S 中，取出 r 个元素按次序排成一列，称为 S 的一个 r-排列，记为：

$$P(n,r),\ P_n^r,\ A_n^r$$

注意，定义里用 r 表示所取元素个数。

当 $r=n$ 时， S 的 n-排列简称为 S 的排列或 n 个元素的全排列。

从 n 个元素中取 r 个元素的排列的典型例子是从 n 个**不同**颜色的球中，取出 r 个，放入 r 个**不同**的盒子里，每盒 1 个，盒子编号就映射成队列位置。显然第 1 个盒子有 n 种球可选，第 2 个盒子有 $n-1$ 种球可选，……，第 i 个盒子有 $n-(i-1)$ 种球可选，……，第 r 个盒子有 $n-(r-1)$ 个球可选。

可以得出

$$A_n^r = n(n-1)\cdots[n-(i-1)]\cdots[n-(r-1)],\quad r\le n,\quad n,r\in\mathbb{Z}^+$$

定义阶乘 factorial：

$$\begin{aligned} n! &= n\times(n-1)\times(n-2)\cdots 2\times 1\\ 0! &= 1 \end{aligned}$$

排列组合是没会有 0 的情况的，但为了数学定义和计算的完备性，要考虑 0 等情况。其实负数的阶成也有定义，不过在组合数学里没意义，在此不考虑。后面还会遇到类似形式的定义。那么：

$$\begin{aligned} A_n^r &= \frac{n!}{(n-r)!},\quad 0\le r\le n\\ A_n^0 &= 1\\ A_n^n &= n! \end{aligned}$$

排列还分为**直线排列**和**圆排列**。直线排列就是我们常说的排列，圆排列是指把取出的元素排成一个圆形，

上面定义的是**直排列**，**圆排列**定义为取出的元素排成一个圆圈，等于是让直线排列的首尾相连接。从 n 个元素中取出 r 个构成的圆 r-排列数为：

$$A_n^r/r,\quad (1\le r\le n)$$

因为 1 个圆排列从任一位置断开都是一个不同的直排列，也就是说 r 个直排列对应 1 个圆排列，所以要在直排列基础上除以 r. 注意，是除不是乘。特殊的，n 个元素的全圆排列数为 $(n-1)!$.

圆排列还有一种情况是，可以翻转圆排列，如 n 个不同颜色的珠子串成一条项链。一条项链的一面是一个普通圆排列，但翻转这个项链，它的另一面是一个新有圆排列，所以每个可翻转圆排列对应 2 个普通圆排列。因此，**可翻转圆排列**数为

$$\frac{A_n^r}{2 \cdot n}$$

4.4 Combination of Sets

上节讨论了单重集的排列，这节讲单重集的组合。排列和组合的主要区别是**不同**元素是否有**次序**。组合取出元素**堆**在一起，而排列在组合的基础上对取出的元素堆进行排队或入盒。

一旦取出，便不再区分组合堆内球的不同。如果取出多个堆，堆间区别是球的个数：堆内无序，堆间也无序。后面多次用到这个原则。

从 S 中取 r 个元素而不进行排序，称为 S 的一个 r-组合，实际就是生成一个 S 的 r 元素子集，可以看成是取出元素堆在一起，没有顺序：组合即组堆，也即生成子集。

组合的对应彩色球问题是从 n 个色彩**不同**的小球中取出 r 个（堆在一起），此时没有放入盒子的操作：只取不入。如果一定要有入盒操作，那么所有的盒子相同，没有编号区分，此时入盒与否没有意义。 r -组合数记为

$$C(n,r),\ C_n^r,\ \binom{n}{r}$$

很显然，一个 r -组合可以生成 $r!$ 个排列，所以:

$$\begin{aligned}
&C_n^r = \frac{A_n^r}{r!} = \frac{n!}{(n-r)!\,r!}, \quad r,\, n \in \mathbb{Z}_0^+ : r \in [0,n] \\
&C_n^r = 0, \quad r > n \\
&C_n^0 = 1 \\
&C_n^n = 1 \\
&C_0^0 = 1
\end{aligned}$$

从上面方程中的特例可以看出，完备性定义对数学计算的意义。阶乘，排列数，组合数的完备性定义主要考虑 $r > n$ 和 $n,\, r = 0$ 的情况。一般地，我们只考虑 $0 \le r \le n$ 的情况。对于 $r,\, n < 0$ 的情况不在排列组合的讨论范围。在其它计算领域即便碰到，也不难，只要严格按照阶乘的定义计算即可。

由公式:

$$A_n^r = C_n^r\, r!, \quad r,\, n \in \mathbb{Z}_0^+ : r \in [0,n]$$

可以得出，除原始定义外，排列操作可看作先取组合，得到一堆元素，再对此堆列队。可以说**排列操作暗含了组合子问题**。简而言之，1 个组合对应 $r!$ 个排列，排列数是组合数的 $r!$ 倍。后面更复杂的分配分组问题也遵循此规律。

4.4.1 Combination Formulas

组合数的计算非常重要，因为组合公式是一个很重要的数学工具，如多项式的系数和组合数紧密相联。本小节着重讲组合数的几个公式。

组合数和排列数计算都化成阶乘的计算。不过，当参数很大时，计算阶乘不是件容易的事。但从组合数的原始定义可知：取 r 个元素得到一个子集，余下的就是 $n-r$ 元素的补集，一一对应。也就是说，每取一个 $(n-r)$-组合就得到一个 r -组合：

$$C_n^r = C_n^{n-r}, \quad r, n \in \mathbb{Z}_0^+ : r \in [0, n]$$

当 r 很大时，$n-r$ 就很小，便于计算。我们还可以想办法降级阶数，让其变得更小：

$$C_n^r = C_{n-1}^r + C_{n-1}^{r-1}$$

由此公式可得出 杨辉三角形，也称 $Pascal$ 三角形。

有趣的是让 r 遍历 0 到 n ，可以看出所有的组合数之和就是集合 S 的幂集的元素个数：

$$\sum_{r=0}^{n} \binom{n}{r} = 2^n$$

此公式可以用二项式证明：

$$(x+y)^n = \sum_{r=0}^{n} C_n^r x^r y^{n-r}$$

令 $x, y = 1$ 即可。

更一般地，组合公式的证明**双重计数 double counting** 法：从不同（一般 2 种）角度对集合计数。更多关于双重计数，看 Double Counting 和 Mod-03 Lec-17 Double counting - Part (2) .

如上面的组合数求和公式，左边表示利用加法原则，以了集元素个数 r 来对 S 的幂集计数。要证明，我们以另外一个角度来计数：利用乘法原则，针对一个元素是否加入子集来计数。$\forall a_i, 1 \le i \le n$ 要么在某个子集里，要么不在，只有两种可能。对 a_i 遍历， a_1 有两种可能， a_2 有两种可能，……，a_n 有两种可能，所以 S 有 $2 \times 2 \times \cdots \times 2 = 2^n$ 个子集。

4.4.2 Application of Combination

现有 n 个管理员管理某保密装置（Minimum number of locks and keys)。要求任何 $\leq r$ 个管理员都打不开该装置，至少需 $r+1$ 个。假设该装置有 s 把钥匙，给每个管理员分配其中的 t 把。问题是已知 $n,\ r,\ r \leq n$, 求 $s,\ t,\ t \leq s$. 并进一步给出该装置的钥匙分配方案。

设管理员集合为:

$$M = \{\, m_1,\, m_2,\, \cdots,\, m_n \,\},\ r \leq n$$

钥匙集合为:

$$K = \{\, k_1,\, k_2,\, \cdots,\, k_s \,\},\ t \leq s$$

先求钥匙数 s. 由描述知，管理员集合 M 的任一 r-组合（管理员子集）都不能凑齐 s 把钥匙，可知任一 r-组合至少还缺 1 把钥匙。显然，任两个 r-组合的钥匙不能完全相同，也即不能缺同一把钥匙，否则 $2r$ 个管理员还因缺一个把钥匙而打不开保密装置，而且这样的分配方案没有意义。 M 的 r-组合数是 C_n^r. 所以:

$$s \geq C_n^r$$

从上分析可看出， M 的所有 r 元素子集到 K 的一个单射：每个 r 元素子集的像是它所缺失的那把钥匙。但它不一定是满射，因为 $s \gneq C_n^r$ 时，额外的钥匙会被所有的 r-组合所覆盖，这样的钥匙没的原像。其中等号成立的条件是每个 r-组合刚好缺一把钥匙，此时构成一个满射。

再求每个管理员分得的钥匙数 t. M 的任一 $(r+1)$ 元素子集都能打开装置，说明其中任一管理员可补齐剩下 r 个管理员所缺的那把钥匙。一个管理员 m_l 所参与的 $(r+1)$-组合有 C_{n-1}^r 个，所以:

$$t \geq C_{n-1}^r$$

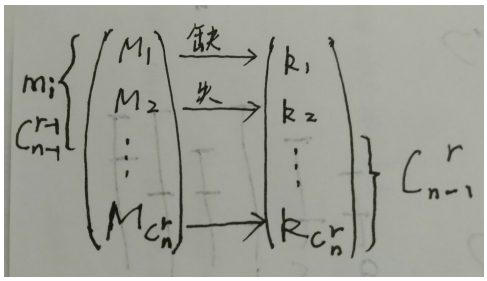
在分析钥匙的分发方案之前，让我们回顾下上面的杨辉三角形，本题已出现公式里两个分项 C_n^r 和 C_{n-1}^r, 只不过后者的意义稍变了。

s 取最小值 C_n^r 时，我们列出 M 的所有 r 元素子集到 K 的满射:

$$f(M_i) = k_j$$

$$\{\, M_i \mid M_i \text{ is a set of size } r,\ i = 1, \cdots, C_n^r \,\} \xrightarrow{\text{缺失}} K$$

显然这样的映射有很多种，我们只需选其一，如下图所示:



对于某个管理员 m_l, 他所在 r 元素子集有 C_{n-1}^{r-1} 个，到此，杨辉三角形公式里三个分项全部出现。很显然这管理员也缺失了这些 r 元素子集对应的钥匙像，所以该管理员应分得剩下的 C_{n-1}^{r} 个钥匙像。针对所有管理员进行类似分配即可。

前面提过，当 s, t 的不等式不取等号时，此映射不是满射，有钥匙没有原像，此时要求所有的 r -组合都包含这样的钥匙。

现假设 $s = C_n^r + 1$, 此时的分配方案只需在原基础上稍加修改即可：多出的这枚新钥匙再分配给每个管理员。这样 $t = C_{n-1}^r + 1$. 此设计仍满足保密要求。我们会发现这样的额外钥匙是没有必要的。每个管理员都多了 1 枚同样的钥匙，起不到额外的安全作用。

分配方案的关键是根据 s 值列出 r 元素子集到钥匙的映射。现给出一个实际的例子。如果 $n = 5, r = 3$, 则 $s = C_5^3 = 10, t = C_4^3 = 4$. 下面给出一个映射。多出的一个钥匙（编号 11）是为了说明映射不一定是满射。

$$
\begin{aligned}
123 &\xrightarrow{\text{缺}} 1\\
124 &\longrightarrow 2\\
125 &\longrightarrow 3\\
134 &\longrightarrow 4\\
135 &\longrightarrow 5\\
145 &\longrightarrow 6\\
234 &\longrightarrow 7\\
235 &\longrightarrow 8\\
245 &\longrightarrow 9\\
345 &\longrightarrow 10\\
&\qquad\quad 11
\end{aligned}
$$

每个管理员参与了 $C_{5-1}^{3-1} = 6$ 个 3 元素子集，对应到映射图上，有 6 行。不失一般性，管理员 m_1, 参与了前 6 行，缺失前 6 枚钥匙，所以分得后 4 枚钥匙 7, 8, 9, 10. 依此类推，我们可以得出如下分配方案。注意，该方案考虑到了第 11 枚钥匙。

Managers \ Keys	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	1	1	1	1	1
2	0	0	0	1	1	1	0	0	0	1	1
3	0	1	1	0	0	1	0	0	1	0	1
4	1	0	1	0	1	0	0	1	0	0	1
5	1	1	0	1	0	0	1	0	0	0	1

Table 4.1: Safe Device

4.5 Permutation of Multiset

排列组合还会考虑所取元素元素是否**重复**，从而生成（多）重排列和（多）重组合。

前面两节介绍了单重集的排列组合，对应地，多重集也有排列组合，定义和单重集一样，唯一的区别是取出的元素可能有重复。

设多重集 M 为:

$$M = \{\, n_1 \cdot a_1,\ n_2 \cdot a_2,\ \ldots,\ n_k \cdot a_k \,\}, \quad \forall i = 1, 2, \cdots, k,\ r \le n_i$$

或:

$$M = \{\, \infty \cdot a_i,\ \infty \cdot a_2,\ \ldots,\ \infty \cdot a_k \,\}$$

由于每种元素的个数超过所取数，所以排列里 r 个元素可能全部来自某一种。

多重集 M 的 r-排列数是:

$$k^r$$

这是很容易理解的，因为队中每个位置都有 k 种可能。

多重集 r-排列对应的彩球模型可以看成**可放回**（可重复）地对单重集取球、排队或入盒。单重集里每个元素可无限重复使用，标记好队列位置后又放回球堆里。当然按照多重集的定义，可以看成是对多重集取球、排队或入盒。如不多于四位的三进制数的个数为 3^4. 对应的多重集是 $M = \{\, \infty \cdot 0, \infty \cdot 1, \infty \cdot 2 \,\}$.

如果 $r = n = \sum_{i=1}^{k} n_i$, 则称为 M 的全排列。多重集全排列（也即 n-排列）是对**有限集** M 的**所有**元素（如彩球）进行操作。具体来说，就是对全部 n 个 k 种颜色的球进行排队。

显然此时 r 大于所有的 n_i. M 的全排列数为:

$$\frac{n!}{n_1!\, n_2! \cdots n_k!}$$

全排列的证明可以先从直觉来分析。$n!$ 表示所有元素的全排列，但是 a_i 在队列重复了 n_i 次，这此重复实际只表示一种队列，所以除以 $n_i!$. 如果某个 n_i 等于 1, 那么在被除式中是 1!.

有趣的是，实际证明中，用的是组合思路。第 1 种元素 a_1 要占据队列里的 n_1 个位置，所以有 $C_n^{n_1}$ 种可能，第二种元素要占据剩下 $n-n_1$ 个位置中的 n_2 个, 所以有 $C_{n-n_1}^{n_2}$ 种可能, ……, 依此类推, 最后一种元素有 $C_{n_k}^{n_k}=1$ 种可能:

$$C_n^{n_1}\, C_{n-n_1}^{n_2} \cdots C_{n_k}^{n_k} = \frac{n!}{n_1!\, n_2! \cdots n_k!}$$

从证明过程可知，当 $k=2$ 时，多得集的全排列数在数值上等于单重集的组合数 $C_n^{n_1} = C_n^{n_2} = n!/(n_1!\, n_2!)$. 如用两面红旗，三面黄旗依次悬挂在一根旗杆上，问可以组成多少种不同的标志？答案是 $5!/(2!\, 3!) = 10$.

下面看一个棋盘的例子。在一个 $k \times k$ 的棋盘上放 k 个车，使得任意两个车之间不能互吃，有多少种方法？棋盘上车全是相同的，没有区别。要想车不不互吃，任意两个车的行列值都不同。每行一个车 $a_{1\,j_1}$, $a_{2\,j_2} \cdots a_{k\,j_k}$, 只需给不同行的车选列即可，所以方法数是 $k!$.

假设是 k 个不同（色）的车呢？保持上面排列不变，现对车的颜色进行调换，有 $k!$ 种，所以方法数是 $k!\, k!$. 实际是对车所在的行进行全排列，也即行和列都要全排列。行的全排列负责车的不同，而列的全排列负责车不互吃。

假设是 n_1 个红车，n_2 个蓝车，……，n_k 个黄车，总共 n 个呢？先解决车在行上的全排列：$\frac{n!}{n_1!\, n_2! \cdots n_k!}$. 再针对不同的列全排列 $n!$. 方法数是:

$$\frac{n!}{n_1!\, n_2! \cdots n_k!} \cdot n!$$

多重集 r-排列要求每种元素个数不少于 r, 若 $\exists t, n_t < r$, 那么情问变复杂了。这时没有公式计算，要具体针对 n_t 列举分析。如第 t 种元素出 $1, 2, \cdots, t$ 个。

例：9 个元素的多重集 $S = 3\cdot a, 2\cdot b, 4\cdot c$ 的 8-排列数为多少？此例中，$r=8$ 大于 n_i, 所以不能套用公式。$n=9$ 只比 r 大 1, 所以列队中，a, b, c 之一少出一个元素。如少一个 a 排列数是 $\frac{8!}{2!\,2!\,4!}$, 同理可算另两项，再用加法原则:

$$\frac{8!}{2!\,2!\,4!} + \frac{8!}{3!\,1!\,4!} + \frac{8!}{3!\,2!\,3!}$$

4.6 Combination of Multiset

多重集 S 的含有 r 个元素的子多重集就叫做 S 的 r-组合，或表述为取出 r 个元素的堆。和多重集的排列比少了列队操作。

关于多重集的排列问题可以小结如下：

设$S=\{n_1.a_1,n_2.a_2,\ldots,n_k.a_k\}$,且$n=n_1+n_2+\ldots+n_k$，则$S$的$r$-排列数$N$满足：

⑴ 若$r>n$，则 $N=0$;

⑵ 若$r=n$，则 $N=\dfrac{n!}{n_1!n_2!\cdots n_k!}$；

⑶ 若$r<n$，且对一切$i=1,2,\ldots,k$有$n_i\geq r$，则S的r-排列数是k^r;

⑷ 若$r<n$，且存在某个$n_i<r$，则对N没有一般的求解公式，解法将在后面讨论。

Figure 4.1: Summary on Multiset Permutation

同多重集的组合样，假设每种元素个数都不小于 r. 方法数即 k 元线性不定方程:

$$r = x_1 + x_2 + \cdots + x_k, \quad x_i \in \mathbb{Z}_0^+,\ i = 1, 2 \cdots, k$$

的非负整数的解数。非负是表明可能某 x_t 为 0, 此时元素 a_t 没有出现在组合堆里。

计算用 隔板法或 插空法。

可以看成这样，有 r 个 1 在一行上，占有 r 位置，包含首尾共有 $r+1$ 个空位，插上 $k-1$ 个板子。第 i 板子前面的 1 个数是 x_i 的解，第 $k-1$ 个板子后面的 1 个数是 x_k 的解。如果某个板 k_t 前没有 1 而是另一个板（两个板子处在同一空位），则 x_t 解为 0. 如从

$$|\,1\,|\,|\,1\ 1\ \cdots\ 1\ 1\,|$$

看出:

$$x_1 = 0,\ x_2 = 1,\ x_3 = 0,\ x_k = 0$$

非负正整数解的思路是：r 个 1 的位置和 $k-1$ 的板位置共 $r+(k-1)$ 个位置，从中取 r 个作为 1 的位置或取 $k-1$ 个作为板的位置:

$$C_{r+k-1}^{r} \quad \text{or} \quad C_{r+k-1}^{k-1}$$

不考虑多重集组合，单就方程本身来说，如果是求正整数解呢？

$$r = x_1 + x_2 + \cdots + x_k, \quad x_i \in \mathbb{Z}^+,\ i = 1, 2 \cdots, k$$

正整数解暗含了 $r \geq k$, 非负解则无此限制。r 个 1 形中间有 $r-1$ 个空档，插入 $k-1$ 个板，且每个空档最多一个板，则解数:

$$C_{r-1}^{k-1}$$

这比非负情况的简单。

实际非负整数解和正整数解之间可以互相转换，形成一个满射，而解个数不变。下面是把非负解转成正整数解的方法，即变量加 1:

$$r + k = (x_1 + 1) + (x_2 + 1) + \cdots + (x_k + 1), \quad x_i \in \mathbb{Z}_0^+,\ i = 1, 2 \cdots, k$$

新方程的解和原方程的解一一对应 (满射：减一)，这种变换思路很重要。新方程的解是正整数解：

$$C_{r+k-1}^{k-1}$$

把正整数解转成非负解，是把变量减 1. 利用转换原理，我们把方程：

$$x_1 + x_2 + x_3 + x_4 = 20, \quad x_1 \geq 3, x_2 \geq 1, x_3 \geq 0, x_4 \geq 5$$

转成非负解形式：

$$(x_1-3)+(x_2-1)+(x_3-0)+(x_4-5) = 11, \quad x_1 \geq 3, x_2 \geq 1, x_3 \geq 0, x_4 \geq 5$$

或正整数解形式：

$$(x_1-2)+(x_2-0)+[x_3-(-1)]+(x_4-4) = 15, \quad x_1 \geq 3, x_2 \geq 1, x_3 \geq 0, x_4 \geq 5$$

后用插板法。

总结：

- 正整数：插板不相临；从中间空槽取 $k-1$.
- 非岁整数：插板可相监；从位数取 $k-1$.

4.7 Polynomial

这节谈下多项式和多重集排列、组合之间的联系。多项式：

$$(x_1 + x_2 + \cdots + x_k)^n$$

的 展开式 可写成：

$$\sum_{n_1+n_2+\cdots+n_k=n} \frac{n!}{n_1!n_2!\cdots n_k!} x_1^{n_1} x_2^{n_2} \cdots x_k^{n_k}, \quad n_i \in \mathbb{Z}_0^+ : n_i \in [0, n],\ i = 0, 1, \cdots, k$$

由此可看出，多项式的每项的系数是一个多重集的全排列数 $\frac{n!}{n_1!n_2!\cdots n_k!}$. 还可算出展开式的项数是多重集组合数 $\binom{n+k-1}{n}$.

如果原多项式里 x_i 前还带有系数如 a_i:

$$(a_1x_1 + a_2x_2 + \cdots + a_kx_k)^n$$

则情展开式的系数在多重集全排列数基础上还有乘以对应的原始系数。

如果问多项式的系数之和是多少，只需给所有 x_i 赋值 1 即可:

$$(a_1 + a_2 + \cdots + a_k)^n$$

对于所有 $a_i = 1$ 的情况，则系数和是 k^n.

4.8 Grouping and Distribution

下面是先说分组和分配问题，方法数和多重集全排列数有关。

4.8.1 Distribution as Injection

将 $n = n_1 + n_2 + \cdots + n_k$ 个 **不同**的球 a_i （单重集）放到 k 个 **不同**的对应盒子 b_i 里（单重集)。b_1 放 n_1 个，b_2 放 n_2 个，……，b_k 放 n_k 个。如果 $k = n$, 它就变成单重集的全排列，此时 $n_i = 1$. 简单描述：把 n 个不同色球分成 k 堆 n_i, 依次分配给不同盒子 b_i.

此模型称为**单重集的定向分配**问题。分配强调是分给**不同**的盒子，而定向是说 n_i 映射到 b_i, 每个盒子所分得的**数量固定**，即每堆的球数已经限定。入盒操作相当于组球堆贴标签。

计算过程和多重集全排列的类似,选取 n_1 个 $C_n^{n_1}$, 再在剩下球里取 n_2 个 $C_{n-n_1}^{n_2}$, ……

这个取堆顺序是 $n_1, n_2, \cdots, n_k$, 实际按 i 的任何一种排列顺序来取都可以，只要所取之堆放入对应的盒子子即可。如第一次取 n_5 个, 相当于给盒子 b_5 取球，那么取出后就放入盒子 b_5.

计算如下:

$$C_n^{n_1} C_{n-n_1}^{n_2} \cdots C_{n_k}^{n_k} = \frac{n!}{n_1! n_2! \cdots n_k!}$$

此计算利用了乘法原则。

最后，多重集全排列计算过程是给每种色球找队列位置（共 n 个)，而上面的模型是给每个盒子（共 k 个）选色球。此模型里彩球集合和空盒集合是单重集，但放球后，盒子集合是多重集。每个盒子代表一种元素，里面的球数代表这种元素的个数。

4.8.2 Grouping

定向分配把球堆分给不同但固定的盒子，如取球放入 k 个的相同的盒子里(盒子没有编号)，此模型称为 **单重集的分组**问题。

由于盒子相同，所以有没有入盒操作对结果不影响，所以入盒是一个空操作。所以有的题里，只提到分组，没有入盒操作（如分派给谁)。单重集分组可以看成把取出球堆也堆在一起，形成一个大堆，大堆内元素是小球堆：分组即分堆。

不失一般性，假设按 $n_1, n_2, \cdots, n_k$ 的顺序取堆，如果就此打住，那就成了定向分配。到底区别哪？没有**去重**！

球堆的区别是球个数 n_i 值的大小（球已取完，不再考虑其不同)，可能某些堆的球数相等，这些堆算作是相同的堆。

不失一般性，假设 $n_1 = n_2 = n_3$ 这 3 个堆球数相等，它们之间先取谁后取谁没有变化，同理 $n_7 = n_9$ 这 2 个堆的先后也不改变分组操作（没有特殊说明，后面都据此假设)。所以要除掉重复：

$$\frac{C_n^{n_1} C_{n-n_1}^{n_2} \cdots C_{n_k}^{n_k}}{3!\,2!}$$

对所有的相同数值做类似去重除法即可。做题时，尽量把相同的 n_i 写在一起，好去识别重复。

我们有理由把

$$\{n_1, n_2, \cdots, n_k\}$$

看成多重集：这 k 个值分成 l 种，每种里有 $t_j,\ j = 1, 2, \cdots, l$ 个元素，有 $\sum_{j=1}^{l} t_j = k$. 拿上面的例子来说：

$$\{3 \cdot n_1,\ 1 \cdot n_4,\ 1 \cdot n_5,\ 1 \cdot n_6,\ 2 \cdot n_7,\ \cdots,\ 1 \cdot n_k\}$$

所以去重就是除以多重集每种元素个数的阶乘 $t_1!\, t_2! \cdots t_j!$, 所以标准表达是：

$$\frac{C_n^{n_1} C_{n-n_1}^{n_2} \cdots C_{n_k}^{n_k}}{t_1!\, t_2! \cdots t_j!}$$

一个特例是，所有 n_i 全相等，表示是平均分组，所以：

$$\frac{C_n^{n_1} C_{n-n_1}^{n_2} \cdots C_{n_k}^{n_k}}{k!} = \frac{n!}{k!\,[(\frac{n}{k})!]^k}$$

再回过头看定向分配，发现可以分解成分组、定向分配 2 步：

$$\frac{C_n^{n_1} C_{n-n_1}^{n_2} \cdots C_{n_k}^{n_k}}{3!\,2!} \cdot 3!\,2!$$

先把不同球数的堆放入对应的盒子。后面乘以 3! 表示 $n_1 = n_2 = n_3$ 这 3 个堆和 b_1, b_2, b_3 间可任意分派。乘以 2! 也是同样的道理。

一句话：**分配问题暗含了分组子问题**。

4.8.3 Distribution without Injection

有定向就有不定向，不定向分配也把球堆分派给不同的盒子，但是没有固定分配关系：球堆 n_i 和盒子 b_j 间没有固定映射关系。把 n 个不同色球分成 k 堆 n_i, 分配给不同盒子 b_j. 此模型称为 **单重集的不定向分配**问题。

始终记住，盒子的不同、盒子编号代表队列位置。不定向分配分解成分组、不定向分派 2 步：

$$\frac{C_n^{n_1} C_{n-n_1}^{n_2} \cdots C_{n_k}^{n_k}}{3!\,2!} \cdot k!$$

由于分组里已考虑到重复问题，后面的排列就不再考虑 $3!\,2!$ 重复，而应把 $n_1 = n_2 = n_3$ 看作是不同的堆，否则就会多次去重。

不定向分配还可以看成在定向分配基础上对 k 个球堆（数值 n_i）进行全排列操作，方法数是：

$$C_n^{n_1} C_{n-n_1}^{n_2} \cdots C_{n_k}^{n_k} \cdot \text{球堆全排队数}$$

在前节分组问题说到，堆的球数 n_i 是个多重集，所以我们乘以多重集的全排列数：

$$\begin{aligned} C_n^{n_1} C_{n-n_1}^{n_2} \cdots C_{n_k}^{n_k} \cdot \frac{k!}{3!\,2!} &= \frac{C_n^{n_1} C_{n-n_1}^{n_2} \cdots C_{n_k}^{n_k}}{3!\,2!} \cdot k! \\ &= \frac{n!}{3!\,2! \cdot n_1!\,n_2! \cdots n_k!} \cdot k! \end{aligned}$$

公式的推导过程也说明了不同的解题思路。

计算过程总结：

1. 球一旦取出，组合计数完毕，就不再考虑球的不同。
2. 取出的球堆区别于堆内球的个数。
3. 可能某些堆球数相等，这些球堆看作是相同的堆。其实是一个多重集。

4.8.4 Distribution of Multiset

上面的问题里，单重集每个球不同，如果所有球全相同呢？若像单重集分组分配样规定 n_i, 则堆数确定，分组问题固定，只有 1 种方法。定向分配问题：n 个相同的球分成 k 组 n_i 定向派给不同盒子 b_i，也只有 1 种方法。不定向分配数是多重集的全排列如 $\frac{k!}{3!\,2!}$.

所以一般不规定堆内球数，如把 n 个同色球放入 k 个不同的盒子里，盒子不为空。其实这个问题可化成求 k 元线性不定方程的正整数解。方法数是：

$$C_{n-1}^{k-1}$$

此时小球可看作是只有 1 种元素的多重集 $M = \{ n \cdot a_1 \}$. 此模型可称为含 1 种元素的**多重集定向分配**问题。

如果盒子可以为空，则化成求方程的非负整数解。

含一种元素的**多重集分组和不定向分配**问题没有统一解法，因为分组和不定向要考虑去重问题，但是去重的前担是要知道 n_i 数值。所以只能一一列举出方程解，再考虑每种解的去重，无法直接写出计数公式。

4.9 Summary

分组分配总结：

1. 单重集分组：固序取堆、去 n_i 重。
2. 单重集定向分配：固序取堆。
3. 单重集不定向分配：固序取堆、去 n_i 重、排列。
4. 多重集定向分配：插板法、解方程。
5. 多重集分组和不定向分配：没有公式，只可一一列举。

Chapter 5

卡特兰数

5.1 卡特兰公式

卡特兰数 Catalan Number 在许多算法计数问题中都有应用，如出栈计数，二叉树形态计数等。卡特兰数也叫明安图数。因为最提出这数的是中国明代数学家明安图。但因为近代以来，科学技术主要由西方科学家发起，所以很多发现用西方人物命名。

明安图数其实是一个组合数：

$$C_n = \frac{C_{2n}^n}{n+1} = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{n!(n+1)!}$$

卡特兰数 C_n 表示对 n 个元素的某种顺序计数。注意这里符号 C_n 不是前面的 n 个元素的组合数里的 C_n^r.

卡特兰数可以写成另外一种形式：

$$\begin{aligned}
C_n &= \frac{C_{2n}^n}{n+1} \\
&= \frac{1}{n+1}\frac{(2n)!}{n!\,n!} \\
&= \frac{1}{n(n+1)}\frac{(2n)!}{(n-1)!\,n!} = (\frac{1}{n} - \frac{1}{n+1})\frac{(2n)!}{(n-1)!\,n!} \\
&= \frac{1}{n}\frac{(2n)!}{(n-1)!\,n!} - \frac{1}{n+1}\frac{(2n)!}{(n-1)!\,n!} = \frac{(2n)!}{n!\,n!} - \frac{(2n)!}{(n-1)!\,(n+1)!} \\
&= C_{2n}^n - C_{2n}^{n+1}
\end{aligned}$$

请注意此化简过程用到

$$\frac{1}{n\,(n+1)} = \frac{1}{n} - \frac{1}{n+1}$$

在处理组合数间题时，$\frac{1}{n-1}, \frac{1}{n}, \frac{1}{n+1}$ 经常被用到来化简。这种倒数可以和阶成结合，组成新的组合数。

上面说的是卡特兰数结果，卡特兰数的递推关系式是:

$$\begin{aligned} C_n &= \frac{C_{2n}^n}{n+1} \\ &= \sum_{k=1}^{n} C_{k-1}\, C_{n-k} \\ &= C_0 C_{n-1} + C_1 C_{n-2} + \cdots + C_{n-1} C_0 \\ &= \sum_{k=0}^{n-1} C_k C_{n-1-k} \end{aligned}$$

如何由这个递推关系得到结果式，要用到产生式或叫母函数 Generating Function:

$$g(x) = C_0 x^0 + C_1 x^1 + \cdots + C_{n-1} x^{n-1} + C_n x^n + \cdots$$

关于如何用母函数求系数，细节请参考组合学课件，这里给出计算过程。上面公式里，我们发现每项是子规模的乘积，所以对母函数平方:

$$\begin{aligned} g^2(x) &= (C_0 x^0 + C_1 x^1 + \cdots + C_n x^n + \cdots)(C_0 x^0 + C_1 x^1 + \cdots + C_n x^n + \cdots) \\ &= C_0^2 x^0 + (C_0 C_1 + C_1 C_0) x^1 + \cdots + (C_0 C_n + C_1 C_{n-1} + \cdots + C_n C_0) x^n + \cdots \\ &= C_1 x^0 + C_2 x^1 + \cdots + C_{n+1} x^n + \cdots \end{aligned}$$

对上式乘以 x 得:

$$x \cdot g^2(x) = C_1 x^1 + C_2 x^2 + \cdots + C_n x^n + C_{n+1} x^{n+1} + \cdots$$

对上式加 $C_0 x^0 = 1$ 得:

$$\begin{aligned} 1 + x \cdot g^2(x) &= C_0x^0 + C_1x^1 + C_2x^2 + \cdots + C_nx^n + \cdots \\ &= g(x) \\ x \cdot g^2(x) - g(x) + 1 &= 0 \end{aligned}$$

解上式得：

$$g(x) = \frac{1 \pm \sqrt[2]{1 - 4x}}{2x}$$

通过变换，得出母函数通项关系，去除系数 C_k, 进而解出母函数关于 x 的表达式。$g(x)$ 是最开始的系数是由 C_k 定义的，通过转换后计算出系数，就得到 C_k.

根据二项式的推广：

$$\begin{aligned} (x + y)^\alpha &= \sum_{k=0}^{\infty} \binom{\alpha}{k} x^k y^{\alpha - k},\ a \in \mathbb{R} \\ \binom{\alpha}{k} &= \frac{\alpha(\alpha - 1) \cdots (\alpha - k + 1)}{k!} \end{aligned}$$

由此得：

$$
\begin{aligned}
\sqrt[2]{1-4x} &= [1+(-4x)]^{\frac{1}{2}} \\
&= \sum_{k=0}^{\infty} \binom{1/2}{k} (-4x)^k \\
&= 1 + \sum_{k=1}^{\infty} \binom{1/2}{k} (-4x)^k \\
&= 1 + \sum_{k=1}^{\infty} \frac{\frac{1}{2}\frac{-1}{2}\frac{-3}{2}\cdots\frac{3-2k}{2}}{k!} (-1)^k 2^{2k} x^k \\
&= 1 - \sum_{k=1}^{\infty} \frac{1\,3\cdots(2k-3)}{k!} 2^k x^k \\
&= 1 - \sum_{k=1}^{\infty} \frac{1(2\cdot 1)\,3(2\cdot 2)\cdots(2k-3)[2\cdot(k-1)]}{k!(k-1)!}\, 2x^k \\
&= 1 - \sum_{k=1}^{\infty} \frac{1\,2\,3\,4\,\cdots\,(2k-3)(2k-2)}{k!(k-1)!}\, 2x^k \\
&= 1 - \sum_{k=1}^{\infty} \frac{[2(k-1)]!}{k!(k-1)!}\, 2x^k
\end{aligned}
$$

结合上式得：

$$
\begin{aligned}
g(x) &= \frac{1 \pm \sqrt[2]{1-4x}}{2x} \\
&= \frac{1 \pm [1 - \sum_{k=1}^{\infty} \frac{[2(k-1)]!}{k!(k-1)!}\, 2x^k]}{2x}
\end{aligned}
$$

考虑到定义时，系数 C_k 是正数，所以正负号里选负号：

$$
\begin{aligned}
g(x) &= \frac{1 \pm \sqrt[2]{1-4x}}{2x} \\
&= \frac{1 \pm [1 - \sum_{k=1}^{\infty} \frac{[2(k-1)]!}{k!(k-1)!} 2x^k]}{2x} \\
&= \frac{1 - [1 - \sum_{k=1}^{\infty} \frac{[2(k-1)]!}{k!(k-1)!} 2x^k]}{2x} \\
&= \sum_{k=1}^{\infty} \frac{[2(k-1)]!}{k!(k-1)!} x^{k-1} \\
&= \sum_{k=0}^{\infty} \frac{(2k)!}{k!(k+1)!} x^k
\end{aligned}
$$

通过系数对比，得出 $C_k = \frac{(2k)!}{k!(k+1)!} = \frac{1}{k+1} C_{2k}^k$.

5.2 卡特兰数应用

卡特兰数的来自于应用问题，7 通过元素进出栈顺序数问题，来说明公式推导过程。

长度为 $2n$ 的 Dyck Word（n 个 x 和 n 个 y 组成字符串）数；n 对括号匹配组成合法运算式数；n 个节点组成二叉树方案数；$2n+1$ 个节点的满二叉树数都是 C_n. 特别此 n 个节点的二叉树，添加 $n+1$ 个叶子节点，形成满二叉树，个数都是 C_n.

二叉树前序序列和中序序列的关系。给定某二叉树前序序列或后序序列，求中序序列的可能数，结果也是卡特兰数。相当于以前序序列或后序序列入栈，中序序列出栈。

有 $n \times n$ 的小方格组成的正方形，要求从一个对角走到别一对角，只能横向或竖向行走，不能回头且不能穿过对角线，问有多少种单调路径？如从左下角往右上角行走，单调路径表示每一步只能向右或向上，即向最终方向走。如果不考虑对角线问题，则竖向应走 n 步，横向应走 n 步，共 $2n$ 步，方法数是 C_{2n}^n. 若考虑对角线限制，假假沿在对角线下方走，则说明任何时刻，右向步数不少于向上步数。肯体参考上面提到的进出栈分析。

这些不同问题可以互相转化。如进栈、出栈，Dyck Word 里的 x 和 y, 左、右括号都互相对应。全部用数学描述是 $2n$ 个 ± 1 串。

注意卡特兰数 C_n 里有 n 表示规模，但是实际问题里，n 可能表示的是 n 对元素，而不是 n 个元素。如括号匹配里，n 对括号有 n 个左括号和 n 个右括号。

Part III

数列

5.3 等差数列

我们知道等差数列的前 n 项和是首项加尾项乘以项数除以 2.

$$\begin{aligned}
a_1 &= a_1 \\
&= a_0 + d \\
a_2 &= a_1 + d = a_1 + d \\
a_3 &= a_2 + d = a_1 + 2 \cdot d \\
&\cdots \\
a_n &= a_{n-1} + d \\
&= a_1 + (n-1) \cdot d \\
&= a_0 + n \cdot d
\end{aligned}$$

这里特别提到 a_0. 严格来说，数列下标从 1 开始（表示第一个项），0 不属于数列下标，但有时为了计算方便，要借用 $a_0 = a_1 - d$.

等差数列，任意连续三项中，前后两项之和是中间项的 2 倍，因中间项减等差 d 是前一项，加等差 d 是后一项。

等差数列的前 n 项和:

$$S_n = \frac{(a_1 + a_n) \cdot n}{2}$$

5.4 等差数列幂和

等差数列的幂和是指等差数列前 n 项的 $t\,(t = 1, 2, \cdots)$ 次幂之和:

$$S_t(n) = \sum_{k=1}^{n} a_k^t = a_1^t + a_2^t + \cdots + a_n^t$$

基本思路是升维：借用 $t+1$ 次二项展开式，得到数列相临两项的 $t+1$ 次幂差。一般来说，我们关注的是平方、立方，即 $t = 1, 2$. 后面不加说明，假设 t 为 2, 则 $t + 1 = 3$:

$$\begin{aligned}
(a+b)^3 &= a^3 + 3a^2b + 3ab^2 + b^3 \\
a_{n+1}^3 - a_n^3 &= (a_n + d)^3 - a_n^3 \\
&= 3d \cdot a_n^2 + 3d^2 \cdot a_n + d^3
\end{aligned}$$

对 n 遍历:

$$
\begin{aligned}
a_2^3 - a_1^3 &= 3d \cdot a_1^2 + 3d^2 \cdot a_1 + d^3 \\
a_3^3 - a_2^3 &= 3d \cdot a_2^2 + 3d^2 \cdot a_2 + d^3 \\
&\cdots \\
a_{n+1}^3 - a_n^3 &= 3d \cdot a_n^2 + 3d^2 \cdot a_n + d^3
\end{aligned}
$$

对 n 个式子求和，得出公式：

$$
a_{n+1}^3 - a_1^3 = 3d \cdot \sum_{k=1}^{n} a_k^2 + 3d^2 \cdot \sum_{k=1}^{n} a_k + n \cdot d^3
$$

此式很明显可以直接算出平方和。但此式受限于 a_0, d 没法直接化简，所以我们关键是住思路。其实等差数列前 n 项和可以看成 $t = 1$, 其前 n 项和除了首尾相法加外，还可以借助 $t + 1 = 2$ 次展开式。

下面以前 n 个自然数的平方和为例介绍求解思路（$a_1 = 1, d = 1$）。首先，我们有：

$$
\begin{aligned}
a_{n+1}^3 - a_n^3 &= (n+1)^3 - n^3 \\
&= 3n^2 + 3n + 1
\end{aligned}
$$

对 n 遍历：

$$
\begin{aligned}
2^3 - 1^3 &= 3 \times 1^2 + 3 \times 1 + 1 \\
3^3 - 2^3 &= 3 \times 2^2 + 3 \times 2 + 1 \\
&\cdots \\
n^3 - (n-1)^3 &= 3(n-1)^2 + 3(n-1) + 1 \\
(n+1)^3 - n^3 &= 3n^2 + 3n + 1
\end{aligned}
$$

把这 n 个式子加起来：

$$
\begin{aligned}
(n+1)^3 - 1 &= 3 \cdot \sum_{k=1}^{n} k^2 + 3 \cdot \sum_{k=1}^{n} k + n \\
3 \cdot \sum_{k=1}^{n} k^2 &= (n+1)^3 - 3 \cdot \frac{(1+n) \cdot n}{2} - (n+1) \\
\sum_{k=1}^{n} k^2 &= \frac{n(n+1)(2n+1)}{6}
\end{aligned}
$$

通过公式，我们发现平方和（$t = 2$）与线性和（$t = 1$）的关系：

$$\frac{\sum_{k=1}^{n} k}{\sum_{k=1}^{n} k^2} = \frac{2n+1}{3}$$

对于其它等差数列，我们用同样方法。如计算 $1^2, 4^2, 7^2, \cdots, (3n-2)^2$.

5.5 等差数列的积和

上面讲的是等差数列每项的 t 次幂和，如果求和时，每项不是 t 次幂，而改成相临 t 项积呢？为简便起见，这里假设 $t = 2$, 更高次幂依此类推即可。

$$S(n) = \sum_{k=1}^{n} a_k \cdot a_{k+1} = a_1 \cdot a_2 + a_2 \cdot a_3 + \cdots + a_n \cdot a_{n+1}$$

类似上面方法，思路是对乘积变换，前后间可能消除子项。具体是这样的，升维：把二项积变换三项积：

$$a_n \cdot a_{n+1} = x \cdot a_{n-1}a_n a_{n+1} + y \cdot a_n a_{n+1} a_{n+2}$$

其中 x 和 y 是特定系数，实际上 $-x = y = \frac{1}{3d}$:

$$\begin{aligned} x \cdot a_{n-1}a_n a_{n+1} + y \cdot a_n a_{n+1} a_{n+2} &= x \cdot (a_n - d)a_n a_{n+1} + y \cdot a_n a_{n+1}(a_n + 2d) \\ &= xa_n \cdot a_n a_{n+1} - xd \cdot a_n a_{n+1} + ya_n \cdot a_n a_{n+1} + 2yd \cdot a_n a_{n+1} \\ &= [(x+y)a_n + (2y-x)d] \cdot a_n a_{n+1} \end{aligned}$$

x 和 y 的值应独立于 a_0, d 成立，则：

$$\begin{aligned} (x+y)a_n + 2yd - xd &= 1 \\ x + y &= 0 \\ 2y - x &= 1 \end{aligned}$$

可以算出 $-x = y = \frac{1}{3d}$:

$$a_n a_{n+1} = \frac{1}{3d}(-a_{n-1}a_n a_{n+1} + a_n a_{n+1} a_{n+2})$$

对 n 遍历得：

$$
\begin{aligned}
a_1a_2 &= \frac{1}{3d}(-a_0a_1a_2 + a_1a_2a_3) \\
a_2a_3 &= \frac{1}{3d}(-a_1a_2a_3 + a_2a_3a_4) \\
&\cdots \\
a_n \cdot a_{n+1} &= \frac{1}{3d}(-a_{n-1}a_na_{n+1} + a_na_{n+1}a_{n+2})
\end{aligned}
$$

得出前等差数列的前 n 项积和：

$$
S(n) = \sum_{k=1}^{n} a_k \cdot a_{k+1} = \frac{1}{3d}(a_na_{n+1}a_{n+2} - a_0a_1a_2)
$$

不难发现，此式关键是首尾两个三项积之差。注意这里借用了 a_0. 下面以

$$
S(n) = 1 \times 4 + 4 \times 7 + 7 \times 10 + \cdots + (3n-2)(3n-2+3)
$$

为例。可推出：

$$
\begin{aligned}
(3n-2)(3n-2+3) &= \frac{1}{9}[-(3n-2-3)(3n-2)(3n-2+3) + (3n-2)(3n-2+3)(3n-2+6) \\
&= \frac{1}{9}[-(3n-5)(3n-2)(3n+1) + (3n-2)(3n+1)(3n+4)]
\end{aligned}
$$

每项被变换成加减法，可以通部分抵消，达到求和目的：

$$
\begin{aligned}
1 \times 4 &= \frac{1}{9}[-(-2) \times 1 \times 4 + 1 \times 4 \times 7] \\
4 \times 7 &= \frac{1}{9}[-1 \times 4 \times 7 + 4 \times 7 \times 10] \\
7 \times 10 &= \frac{1}{9}[-4 \times 7 \times 10 + 7 \times 10 \times 13] \\
&\cdots \\
(3n-2)(3n+1) &= \frac{1}{9}[-(3n-5)(3n-2)(3n+1) + (3n-2)(3n+1)(3n+4)]
\end{aligned}
$$

这里 $a_0 = 1 - 3 = -2$. 所有等式相加，得：

$$
S(n) = \frac{1}{9}[(3n-2)(3n+1)(3n+4) + 8]
$$

5.6 幂和与积和的联系

至此等差数列的幂和与积和都可得出。我们发现求幂和与积和的通用思路是**升维**。幂和借用 $t+1$ 次二展开式，而积和借用 $t+1$ 项积。

其实幂和可以用积和的方式计算：

$$\begin{aligned} a_n^2 &= a_n \cdot [(a_n + d) - d] \\ &= a_n \cdot [a_{n+1} - d] \\ &= a_n a_{n+1} - d \cdot a_n \end{aligned}$$

Part IV

Algorithm

Chapter 6

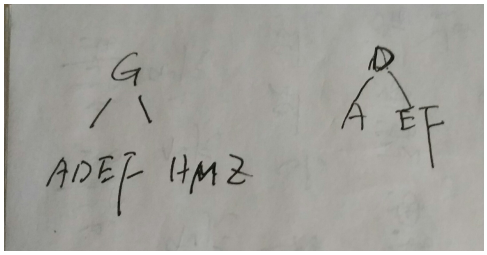
Tree

6.1 前序中序求后序

给出一个二叉树的前序遍历 GDAFEMHZ 和中序遍历 ADEFGHMZ, 求后序遍历。首先，二叉树的前序遍历，中序遍历，后序遍历的区别是根节点是先、中、后访问。问题的关键是：

1. 前序遍历的第一个节点是根节点；
2. 在中序遍历里找到根结点，其左边是左子树节点，右边是右子树节点；
3. 对左子树重复上面两步；
4. 对右子树重复上面两步。
5. 二叉树被求出，从而得到对应的后序遍历。

上面的例子里，从前序遍历知 G 是根节点，从后序遍历知，左子树是 ADEF, 右子树是 HMZ, 此顺序就是左子树的中序遍历顺序。



但还要找到对应的前序顺序。对左子树 ADEF 来说，在前序遍中的顺序是 DAFE，说明左子树的根节点是 D, 从其中序顺序可知，左子树是 A, 右子树是 EF. 重复此过程即可得出二叉树，再得出后序遍历 AEFDHZMG.

同理，已知后序遍历和中序遍历，也可求出前序遍历。唯一的区别是后序遍

历的最后一个节点是根节点，每次找根节点时从最后找。不管是哪种题，中序遍历一定要有！

Chapter 7

栈

7.1 进栈出栈

栈是一种连续据构，数据在栈内连续存储。栈有栈顶和栈底类似链表的首部和尾部，区别是栈在垂直上下方向生长，链表在水平左右向生长。如果需要，我们也可以用水平示意图表示栈，左边是栈底，往右生长，如数组模拟栈就是这种思路。C/C++ 内存中的栈区即是一种栈结构。

给定一有 n 个元素的有序序列 $a_1, a_2, \cdots, a_n$, 对栈有两种基本操作：进栈 push 和出栈 out. 无论是进栈还是出栈，都是对栈顶进行操作。进栈也称压栈，表示保持元素间相对顺序把元素存到栈顶，而出栈则表示取出栈顶的元素。

进栈、出栈组成一个动态操作序列，在 n 个元素依次进栈的过程中，伴随着元素随机出栈。动态操作序列有 $2n$ 个步骤，其中 n 个是进栈操作，另 n 个是出栈操作。进出栈操作结束，栈为空，进栈的 n 个元素亦全部出栈。

进栈顺序事先给定，用下标 i 表示，表示为 $a_i,\ i \in \{1, 2, \cdots, n\}$. 出栈序列是进栈序列的某种排列，可记为 $a_{j_l},\ j_l \in \{1, 2, \cdots, n\},\ l \in \{1, 2, \cdots, n\}$. a_i 进栈操作和 a_{j_l} 出栈操作相间出现。$2n$ 个动态操作序列可记成：

$$c_k,\ k \in \{1, 2, \cdots, 2n\},\ c_k \text{ 是 } a_i \text{ 进栈或 } a_{j_l} \text{ 出栈}$$

可以肯定 c_1 肯定代表 a_1 进栈，c_{2n} 表示某元素出栈。

实际上给定一个完整的出栈序列 a_{j_l} 我们可以还原进出栈操作序列 c_k, 所以通常只需出栈序列 a_{j_l} 即可。给出进栈序列 a_i 在 c_k 中的位置，我们可以推出剩下 n 个位置的出栈序列 a_{j_l}.

7.2 先进后出

进栈出栈遵循**先进后出，后进先出**的原则。对于当前栈内元素，后进栈的压在先进栈的上边，栈顶永远是当前最后一个进栈元素，所以先进栈的肯定比后进栈的后出栈，或说后进栈的肯定比选进栈的先出栈。换句话说，任一时刻，栈的状态决定了当前栈内元素的相对出栈顺序。

总结起来，进出栈序列 c_k 满足：

- a_i 间的相对位置固定，因为进栈顺序事先给定。
- a_i 和 a_{j_l} 符合先进后出。

可以看出，上面说的进栈出栈，不是指所有 n 个元素全部一次性入栈，再一次性出栈。实际过程是进栈、出栈两种操作是交替进行，只要栈不为空，就有可能进行出栈操作。如 a_1, a_2 依次入栈（栈顶到底依次是 a_2, a_1，所以 a_2 肯定比 a_1 先出栈），接着 a_2 立即出栈，a_3 入栈（栈内依次是 a_3, a_1，所以 a_3 肯定比 a_1 先出栈）。

前对对进出栈过程分析的比较清楚，任一时刻的栈内元素符合先进后出原则。那么此原则是如何体现在完整操作序列 c_k 里的呢？

出栈序列里任一元素 a_{j_l}，比其后出且先进的元素倒序。这些元素排在 a_{j_l} 后面（后出栈），而且下标比 j_l 小（先进栈）。它们（包括 a_{j_l} 在内）一定是按下标降序排列！

我们可以据此判断一个出栈序列是否正确。如 $3, 4, 5, 1, 2, 9, 8, 7, 6$ 是出栈序列的下标 j_l，显然这个出栈序列不正确，因为出现了 $3, 1, 2$. 存在某个栈状态，3 压在 1 和 2 上面，栈内元素由上到下依次是 $3, 2, 1$.

为了便于运算，我们一般用下标直接代替元素本身！在验证时，我们遍历 j_l，从其开始，后面比其小的是否全部降序。由上例，$4, 1, 2$ 和 $5, 1, 2$ 都不合法。

7.3 栈深度最小值

给出出栈序列 a_{j_l}，问栈深度最小值是多少。

在进出栈动态操作过程中，元素或进栈或出栈，栈内元素个数不定。栈需容纳未出栈元素。如果每进栈一次，再出栈一次，则栈深度只需 1 即可。如果所有元素全部入栈再出栈，则栈深度最少为 n.

对于一般情况，该如何算呢？其实很简单，只需利用上面的降序原则。遍历出栈序列，对 a_{j_l}，数出从其开始往后，降序元素个数（包含 a_{j_l}），最大值即为栈的最小深度。

7.4 出栈序列数

出栈序列数是指，给出进栈序列 a_i, 有多少种出栈序列 a_{j_l}?

操作序列 c_k 有 2n 个位置，一旦确定了进栈序列 a_i 的 n 个位置，则立马得到出栈序列。也可以选出 n 个位置作为出栈序列，但是出栈序列内部还有先后问题，所以用前一种思路更好。

从 $2n$ 个位置中选取 n 个的方法数是组合数 C_{2n}^n, 但进栈序列的选位方法数比这小。如 a_1 进栈肯定在第 1 位，而且 a_i 肯定不能排在最后 n 个位置。所以最终的方法数肯定是 C_{2n}^n 除以或减去某个数。

如果对进出栈计数，用 $+1$ 表示一次进栈，-1 表示一次出栈，则任何时候栈内元素个数就是这些 ± 1 的和。显然，任一时刻，栈内元素不能是负数个，入栈次数肯定不少于出栈次数，$+1$ 个不小于 -1 个数，± 1 的和 $\geqq 0$.

对应到 c_k, $k \in \{1, 2, \cdots, 2n\}$ 序列，从头至尾扫描（水平示意图），对 k 编历, 对任意 k, 前 k 个位置里，$+1$ 个数不少于 -1 个数。进出栈操作完毕，n 个 $+1$ 和 n 个 -1 总和为 0. 有时为了方便，直接用 $\pm$ 表示即可。有的地方用 $+1$ 和 0 分别表示进栈、出栈，最后的 C_k 是一个二进制数。

总结起来是，**进出栈序列 C_k 的所有前缀子串皆满足 $+1$ 个数不小于（大于等于）-1 个数**。

特别地，当 $k = 1$ 时，前 1 位只能是 a_1 进栈，和加 1 为 1. 当 $k = 2n$ 时，最后 1 位是某元素出栈，和减 1 为 0.

所以我们要在 C_{2n}^n 的基础上排除和为负的情况，也即在栈为空时或为负时，进行出栈操作，也即在遍历 k 的过程中遇到和为负数情况。

假设在遍历 k 的过程中，第 1 次遇到和为 -1 时，有 m 个 -1, 则 $+1$ 为 $m-1$ 次，此时 $k = 2m-1$, $m \in \{1, 2, \cdots, n\}$. 显然 k 是奇数，m 最大值是 n. 特别地，第 k 位是 -1, 前面 $2m-2$ 位中，分别有 $m-1$ 位 $+1$ 和 -1. 后面 $2n-k$ 位里面有 $n-m+1$ 个 $+1$, 剩下 $n-m$ 位是 -1.

对于奇数 k, 非法方法数是不是

$$C_{2m-2}^{m-1} C_{2n-(2m-1)}^{n-(m-1)} = C_{2m-2}^{m-1} C_{2n-(2m-1)}^{n-m}$$

呢？此式表示前 $2m-2$ 个位置里选 $m-1$ 个出来放 $+1$ 表示进栈，剩下的 $2n-(2m-1)$ 个位置里选 $n-(m-1)$ 个放剩下的 $+1$. 从 -1 的角度分析，结果是 $C_{2m-2}^{m-1} C_{2n-(2m-1)}^{n-m}$. 答案是否定得！因为这种选取方法不能保证前面 $k-1$ 位里不出现和为负的情况。

我们可以换个思路。第 1 次遇到和为 -1 时，后面 $2n-k$ 位的和是 $+1$，即进栈次数比出栈多 1. 如果交换后面的 ± 1, 则可以得到一个满射，组合计数不变。交换后面的 ± 1 后，$2n$ 位里面有 $n+1$ 个 -1 和 $n-1$ 个 $+1$, 总和为 -2,

出栈比进栈多 2 次。反过来，总和为 -2 时总能找到第 1 次和为 -1 的情况。所以总的非法进出栈方法数是 $C_{2n}^{n+1} = C_{2n}^{n-1}$, 那么合法进出栈总数为:

$$C_{2n}^{n} - C_{2n}^{n+1} = \frac{C_{2n}^{n}}{n+1}$$

这其实是一个卡特兰数 5.

至此，我们从组合数的角度直接得出进栈数序列数为卡特兰数。进出栈序操作序列，对应对组合数学问题，是在 $2n$ 个位置上，选取 n 个放 $+1$ 或 -1, 要求是对位置 k 遍历，和为非负整数。

7.5 进出栈递推方法

下面我们从递推方法分析进出栈问题。分析算法问题时，我们可以找出问题不同规模之间的联系，具体点就是递推关系。在算法领域，初始条件非常重要。同一公递推公式的因为不同的初始条件而产生不同的数列。

用 f(n) 表示出栈序列数。先看进出栈的小规模问题。明显 $f(1) = 1$, 只有 1 个元素 a_1 时，其进栈再出栈，只有一种可能。对于 $f(2) = 2$ 时:

1. 第一种是 a_1 进，a_1 出，a_2 进，a_2 出: $+-+-$
2. 第二种是 a_1 进，a_2 进，a_2 出，a_1 出: $++--$

当 $n > 2$ 时，我们就要抽象分析了。分析有两个思路。第一种是考虑 n 个元素里，哪个最先出栈，也即出栈序列里，谁是第 1 个元素。第二种是考虑第 1 个元素 a_1 出栈时所在位置，也即 a_1 在出栈序列排第几位。我们先看第一种思路。

7.5.1 最先出栈元素

如果是 a_1 最先出栈，则出栈序列是 $a_1, a_{j_l},\ j_l \in \{2, 3, \cdots, n\}$. 很明显，$c_k$ 前两位是 $+-$, 表示 a_1 进栈后立马出栈。a_1 进栈、出栈后不影响其后元素的进出栈，因为后面 $n-1$ 个元素根本还没入栈，这是一个 $n-1$ 规模的子问题，有 $f(n-1)$ 种可能。

如果是 a_2 最先出栈。此时其前元素 a_1 已入栈并且没出栈。同理 a_2 最先出栈不影响后面元素 $a_i, i \in \{3, 4, \cdots, n\}$ 的出栈序列，因为后面元素根本还没入栈。其后元素有 $f(n-2)$ 种可能出栈序列。其前面的 a_1 穿插于其后元素出栈序列之间。

对于一般情况，$a_k, k \in \{1, 2, \cdots, n\}$ 最先出栈。此时 $a_i, i \in \{1, 2, \cdots, k-1\}$ 已入栈并且没出栈。由前面分析可知，这 $k-1$ 个元素元素下标比 k 小（先

进栈)，但比 a_k 后出栈，在出栈序列里必须是（相对）降序排列（包括 a_k 在内)，即 $a_k, a_{k-1}, \cdots, a_1$. 因为在 a_k 先出栈的条件下，从栈顶到栈底依次是 $a_{k-1}, a_{k-2}, \cdots, a_1$. 特殊的，如果 $k = n$, 则 a_n 最先出栈，说是所有 n 个元素依次全部进栈，然后依次倒序出栈。

对于下标比 k 大的元素而言，a_k 最先出栈不影响 $a_i, i \in \{k+1, k+2, \cdots, n\}$ 出栈。后面元素的出栈是规模为 $n-k$ 的子问题，有 $f(n-k)$ 种可能。

这个一般情况有多少种可能呢?考虑出栈序列 $a_k, a_{j_l},\ j_l \neq k, j_l \in \{1, 2, \cdots, n\}$. 除了第 1 位 a_k, 余下 $n-1$ 位里先选 $k-1$ 个出来按序放 $a_{k-1}, a_{k-2}, \cdots, a_1$. 余下的 $n-k$ 位是 $a_i,\ i \in \{k+1, k+2, \cdots, n\}$ 的出栈序列，对应序列数是 $f(n-k)$. 所以最终是是:

$$C_{n-1}^{k-1} f(n-k) = C_{n-1}^{n-k} f(n-k),\ k \in \{1, 2, \cdots, n\}$$

为保证定义完备性，当 $k = n$ 时，我们定义 $f(n-n) = f(0) = 1$. 所以 $a_i,\ i \in \{1, 2, \cdots, n\}$ 的出栈序列数是:

$$f(n) = \sum_{k=1}^{n} C_{n-1}^{n-k} f(n-k) = \sum_{k=1}^{n} C_{n-1}^{k-1} f(n-k)$$
$$f(1) = f(0) = 1$$

很可惜这个结果是错的！乘法原理要求相乘的两个部分没有交集。上面的分析里，保证了后面 $n-k$ 个元素的出栈相对序列，也保证了前面 $k-1$ 个元素倒序出栈，但当两个部分合在一起时，用乘法原理时，有部分不符合要求。如 $a_k, a_{n-1}, a_2, a_1, a_{n-3}, \cdots$ 这个出栈序列里，a_k 前面的 a_2, a_1 保持了降序，a_k 后面的 a_{n-1}, a_{n-3} 是子问题，但是合在一起时，$a_{n-1}, a_2, a_1, a_{n-3}$ 这 4 个元素不是严格降序。

虽然这个分析错误，但能加强我们对进出栈问题的理解。

7.5.2 a_1 出栈位置

下面从最先入栈的元素 a_1, 也即第 1 个元素出栈序号入手分析问题。设 a_1 第 $k,\ k \in \{1, 2, \cdots, n\}$ 位出栈，则在 a_1 前有 $k-1$ 个元素出栈，a_1 后有 $n-k$ 个元素出栈。

可以肯定，a_1 第 k 位出栈时，栈刚好为空，表明有 k 个元素入栈并完全出栈。那么这 $k-1$ 个元素是哪些呢？a_1 第 1 位入栈后，有 $k-1$ 个元素先入进出栈，a_1 等这些元素出栈后才最后出栈。明显这 $k-1$ 个元素就是 $a_i,\ i \in \{2, 3, \cdots, k-1\}$.

假设 a_l, $k \le l \le n$ 在前 k 个出栈元素中。若 a_l 在第 1 位出栈，据上一小节思路，这 k 个位置显然放不下 $a_l, a_{l-1}, \cdots, a_1$ 这 l 个元素。若 a_l 在第 2 位，则 a_l 和 a_1 间的距离更短了，而且此时第 1 位是比 a_l 更靠后的元素了，同样容不下那么多元素。

以 $k = 2$ 为例, a_1 出栈前要等 1 个元素先出栈。这个元素只能是 a_2. 假假是 a_3, 则出栈序列是 $a_3, a_1, \cdots$, 显然中间的 a_2 没有正确出栈。从另一个角度看，a_3 第 1 个出栈，这个序列没有保证 a_3, a_2, a_1 间相对降序出栈。因此 a_1 等的不可能是 a_2 后的任何元素。

这 k 个元素的进出栈是一个完整的子问题，最先入栈的最后出栈。由于第 k 位确定为 a_1, 则规模为 $k-1$. 方法数是 $f(k-1)$.

在 a_1 后面进出栈的元素是 a_i, $i \in \{k+1, k+2, \cdots, n\}$ 这 $n-k$ 个元素。通过上面分析得，a_1 在第 k 位出栈时，其前 $k-1$ 个元素和后 $n-k$ 个元素的进出栈序列完全独立，互不影响。后面 $n-k$ 个元素的进出栈也是一个完整的子问题，规模为 $n-k$, 方法数是 $f(n-k)$.

根据乘法原理，a_1 第 k 位出栈时，方法数为 $f(k-1)f(n-k)$. 由此 n 个元素的进出栈方法数是:

$$
\begin{aligned}
f(n) &= \sum_{k=1}^{n} f(k-1)\, f(n-k) \\
f(0) &= 1
\end{aligned}
$$

此公式就是卡特兰数 5 的递推公式。不像上节，这节分析没有错误。两个子问题的进出栈序列集合交集为空，符合合乘法原理。

不难发现，进出栈动态操作过程中，一旦栈为空，就得到一个规模更小的完整子问题。进出栈问题，可以分成很多小的子问题，这此子问题问互不影响。特别地，当出栈序列为 $a_1, a_2, \cdots, a_n$ 时，每一步出栈操作都定义了一个子问题，规模为 1, 一共有 n 个子问题. 任意 k, a_k 进栈再出栈时栈为空，构成一个进出栈子问题。

前面分析中，在给出 a_1 出栈得到两个子问题，刚好互逆，并集是进出栈序列的全集。对这两个子问题，我们还可以分成规模更小的子问题。这是从已知结果前提下，一个递归分解过程。

另外，在出栈序列里，若 a_1 所在位置为 k, 可以肯定 a_1 前的元素必须是 a_i, $i \in \{2, 3, \cdots, k-1\}$. 同理, a_1 后的元素肯定是 a_i, $i \in \{k+1, k+2, \cdots, n\}$. 如果出现超出此范围的元素，则出栈序列非法。由此也可以快速排除一个非法的出栈序列。不过，这只是进出栈序列的一个必要非充分条件，不能用来找出正确出栈序列。

7.5.3 递归编程

进出栈问题里，栈的状态可以由两个参数表示，特入栈元素（栈外）个数 n 和栈内元素个数 k, n 和 k 组成一个有序队 (n, k), 我们用符号 $f(n, k)$ 表示进出栈操作序列。下一步可以进栈 $f(n-1, k+1)$, 也可以是出栈 $f(n, k-1)$. 显然 $k = 0$ 时不能出栈。

$f(n, 0)$ 表示初始状态，$f(0, 0)$ 表示操作结束。用此思路，我们可以用递归函数 $f(n, k)$ 列出所有进出栈序列。

7.6 进出栈总结

1. 进出栈序列串，任意前缀子串满足 $+1$ 个数不小于 -1 个数。
2. 出栈序列串，任意 k, 下标比 k 小的并排 a_k 后的元素，其下标降序。
3. 出栈序列串，a_1 出栈时，左边是 $a_i,\ i \in \{1, 2, \cdots, k-1\}$ 子问题, 右边是 $a_i,\ i \in \{k+1, k+2, \cdots, n\}$ 子问题。
4. 进出栈动态过程中，每次栈为空时，得到一个子问题，所有子问题合并是总问题。

Part V

Programming

Chapter 8

C Language

8.1 申明和定义

1. extern 申明 declare 一个变量，只说明存在某个数据类型的变量符号存在（插入到符号表里），而不分配空间。一般来说，在一个文件定义，在另一个要访问该变量的文件里申明（如头文件）。
2. 没有 extern 关键字时，一般就是变量定义。变量定义隐含了申明。定义时要分配内存空间。
3. 函数的声明不需要给出 extern 关键字。

8.2 作用域和生存周期

变量的属性有作用域和生存周期（storage duration）之分。从作用域来说有全局变量和局部变量。从生存周期来说有自动变量（automatic variable）和静态变量（static）。

全局变量在花括号外定义的，默认从定义处开始，到文件结束可见。要想真正实现“全局”的作用域，得在引用前申明变量（函数同理）。局部变量在花括号内部定义，只在花括号内可以被访问，也即局部作用域。花括号可以包裹函数体，也可以是包裹一小段块代码。同名局部变量覆盖 override 全局变量，此时要访问全局变量应用花括号和 extern 关键字。

生存周期和作用域紧密相联。自动变量是指进入作用域时分配内存，而离开时释放内存，变量被销毁。静态变量在进程动行期间一直存在，不受作用域影响。注意静态变量可以在花括号内（代码块或函数）定义！

自动变量和局部变量是同一个概念，只是强调的属性不同罢了，可以称为“局部自动变量”。

静态变量其实是全局变量的缩小版，作用域缩小，不能被 extern 申明引用，所以静态变量在花括号外或文件之外是不可见的。静态变量只在所定义花括号内（局部作用域）或花括号外所定义处至当前文件结尾（文件作用域）范围内有效。同时，静态变量又可以看成是局部自动变量的升级版，主要是生存周期变长，作用域（文件作用域）可能增大。静态变量是非常特殊的可以按作用域分成局部静态变量和文件静态变量。

$$\text{全局变量} \underset{\text{作用域增大}}{\overset{\text{作用域减小}}{\rightleftharpoons}} \text{静态变量} \underset{\text{生存周期变长}}{\overset{\text{生存周期变短}}{\rightleftharpoons}} \text{局部自动变量}$$

下面总结几个要点：

1. 鉴于作用域的重要性，我们一般先定义变量、函数，再引用。否则要先申明，才能引用。
2. 全局、静态变量没有显式初始化时，程序会默认初使化为 0. 初始化（显式或隐式）只会进行一次，即使定义语句多次被进程触及。
3. 局部自动变量没有显式初始化时，初始值不确定，依赖编译器，因为 C 语言对此没有规定。局部自动变量定义时应初使化。
4. 提倡定义全局常量，尽量避免定义全局变量。如果实在需要，用 static 静态变量减小作用域。
5. extern 申明、引用其它文件的全局变量。最好放在头文件里。
6. C++ 对所有变量、对象全部要初始化！

8.3 编译内存布局

C 语言编译后虚拟内存分很多区域。我们这里说的是虚拟内存，假设机器上只有一个程序在运行，占有全部内存。实际中程序执行时所用的内存由操作系统分配在物理内存上：虚拟内存到物理内存的映射！

图 8.1 给出了虚拟内存的安排，简洁地说，就两部分，代码区和数据区。数据区按作用域、生命周期可分堆、栈、常量、全局、静态。对于全局和静态变量，按是否初始化，分为初始化和未初始化。常量肯定算作初始化的。其中的 uninitialized data, 也即 BSS 表示 “block started by symbol”. 总结来看是：堆，栈，常，代，全、静（初始化或未初使化）。

特别的，图中的最高地址处的命令行参数和环境变量也算作是栈，只不过是程序在加载时分配，相当于最早入栈的。其中命令行参数部分是传给 main 函数的 argc 和 argv. 程序运行时还可以指定环境变量，而不是用系统默认的，如语言编码等。如 `LC=zh_CN.GB18030 ./a.out arg1`.

High address	Command-line arguments; Environment variables.	Initial value
	Stack (downward)	
	...	
	Heap (upward)	
	Uninitialized global and static data (bss)	Initialized to zero by exec
	Initialized global and static data	Read from .exe/.out by exec
Low address	(Initialized) Constant data	
	Text/code	

栈	堆	全、静	常	代
数据				代码
未初使化时初值不确定		未初始化	初始化	
依赖编译器		缺省为 0		

Figure 8.1: C Memory Layout

那么编译后的可执行文件如 a.out (Linux 的 ELF 格式) 或.exe 是什么样的结构呢？可执行文件里没有 BSS 段，因为还没有值，只有真正执行时才会有对应的内存。同理栈、堆也不存在，这三部分只对进程有意义。只有常量，初始化过的全局和静态变量。不过可执行文件还包含符号表，段表，库链接表，调试等内容。

堆需手动分配、释放，常用 new/delete 和 malloc/free 操作，由低地址往高地址增长。堆的特点是可根据需求动态分配大小 dynamic allocation on demand. new/malloc 在堆上分配的内存空间是全局有效的。如果不手动 delete/free, 则这些分配到的内存在程序执行过程中一直被其占据，直到程序结束，造成常说的“内存泄露”，导至程序被卡死。所以通常在相同作用域内申请和释放 (如函数内)。

栈是机器自动分配、释放。主要放函数参数，局部变量（自动变量）等。栈上内存主只在函数局部才有效，函数结束会立即被释放。相反，栈的增长方向是由高往低，这样是合理的，如果往同一个方向增长，不便于两个区间的管理。

常量区是指存放常量的地方，如:

```
char * p = "hello, world";
printf("%d", 3);
```

里 p 是自动变量，分配在栈上。但是字符串 “hello, world” 和数字 3 这样的属于量放在常量区。常量区也算作初始化区的一部分。

代码区不用说就是存放放程序代码的地方。

全局变量区是指存放全局变量的地方。静态区是放静态变量的地方。通常这两个区是合并在一起的。而且这个“全静区”是可以分成未初始化区和已初始化区两部分。

程序执行时，先把可执行文件（代码，常量，已初始化全、静变量）加载进内存低地址，再分配未初始化的全、静变量空间。进一步环境变量和程序参数入栈。后面就是在运行时栈和堆的操作了。

说了这么多，我们来看看 Linux ELF 可执行方件的结构:

```
zack@tux ~/workspace/c $ size a.out
 text    data     bss     dec     hex filename
 2391     616      16    3023     bcf a.out
```

Listing 8.1: EFL Layout

这里 text 指代码段大小，data 是已初使化全、静变量和常量段大小，bss 段内数据全零。后面的 dec 和 hex 是前面列的和。

8.4 数组和链表

数组和链表都是线性存储结构，但实现方法不同，通常前者在栈上由机器自动分配，后者在堆上由程序动态分配。实现方法的不同表现出对应的优缺点。

结构 / 优缺点	数组	链表
访问	通过下标随机放问	要遍历链表顺序查找
增删	慢：要先扩容再移动元素	快：只需修改指针指向

Table 8.1: 数组和链表

如果是在中间插入删除（如有序结构），则数组更麻烦，移动的元素更多。

Chapter 9

CPP

Chapter 10

Python Programming

10.1 ABCs

1. Dynamic typing. Contrary to static language, dynamic programming don't have declaration or data type associated.
2. Interpret and execute: source code -> bytecode -> machine code. It is different from C/CPP compilation.
3. Object-oriented Programming (OOP). Everything is object, variables included.
4. Easy but powerful language for scripting and rapid application development.

Dynamic versus Static Static typed programming language do *type checking* (i.e. the process of verifying and enforcing the constraints of types) at *compile-time* as opposed to run-time. Type is associated with compile-time identifiers.

Dynamic typed programming language do *type checking* at *run-time* as opposed to compile-time. Type is associated with run-time values.

```
1 def silly(a):
      if a > 0:
3         print('greater than zero')
      else:
5         print(5 + '3')
7
  silly(1)
9 silly(-1)
```

Listing 10.1: Python Type Checking

Statement `silly(1)` executes perfectly fine and prints 'greater than zero'. Immediately, the next statement `silly(-1)` raises type error:

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

10.2 Terminology

1. the prompt of shell.
2. Procedure-oriented programming (imperative); functional programming (declarative).
   (a) imperative statement vs functional exprenssion/declaration.
   (b) a statement may include an expression (including assignment) or may be just a definition or declaration.
   (c) the key diff is *side effect*.
3. literal constants: number & string

10.3 Execution

To execute Python code, we have basically two forms, namely the *interactive shell* and *source code interpretation*.

To get out of an interactive envrionment, we press `Ctrl-d` or input `exit()`. Please be noted to add the parentheses pair.

10.4 Syntax

Preliminary summaries:

- Comments are any text to the right of a # symbol.
- Literal constants include *numbers* (integers and floats) and *strings*, like `52.3E-4`.
  - Compared to literal constants, Python has *variable* which store values that vary during execution.
  - There is no separate integer type such as *long* or *short*.
  - There is no separate *char* type either.

  - Strings are enclosed with single, double and even triple quotes. Triple quotes can span over multiple lines. Choose one appropriately. For instance, for a string containing both double and single quotes, we use triple quotes on the outer side.

- Apart from numbers and strings, we can define our own variable types as *class*.
- Python starts counting from 0.
- Surround top-level function and class definitions with two blank lines.

10.5 Identifier

Variables are just an example of identifiers. Identifiers are names given to identify something such as variables, function names, class etc.

Identifiers follow the following rules:

- The first character must be a letter of alphabeta (ASCII) or an underscore _.
- The rest of an identifier can consist of letters, underscores, and digits.
- Identifiers are case-sensitive.

Examples of invalid identifiers are:

```
2things, this contains spaces, my-name, <abc
```

`_my_name` is a valid identifier.

10.6 Strings

As mentioned in the above section, strings belong to literal constants that are immutable like that in C/CPP. However, we can construct a new string by different techniques such as the *format* method.

```
# comment to the right of symbol '#'
print('hello, world')

# number and string
age = 20
name = 'jimgray'

# 0 and 1 represents the argument indices
print('{0} is {1} years old when he went to HKUST'.format(name, age))
```

```
print('He was at his {1} years old'.format(name, age))
# give parameter a name
print('we can name the parameters: {name} is {age}'
      'years old when he went to HKUST'.format(name=name, age=age))
print('{0:.3f}'.format(1.0/3))
print('{0:_^11}'.format('hello'))
# formated string (f-string) is an expression evaluated at runtime
print(f'his name is {name} and he is {age} years old.')
print('''this a multi-line string.
The second sentence.''')
```

Listing 10.2: Python Strings

print append newline `\n` to each statement. We can specify the *end* parameter to terminate this behaviour like:

```
print('a', end='')
print('b', end='')
```

10.6.1 Escape Sequence

Although we can enclose a string containing a quotatation mark with double quotes, *backslash* can escape special symbols. For example, `"what's your name"` are equal to `'what\'s your name'`.

To escape the backslash itself, we use double backslash like `\\`.

If we want to specify a two line string, just insert a `\n` symbol. Similarly, we have `\t` for tabular symbol.

You may already know that a bare backslash at the end of line continues the previous element like:

```
"this is the 1st string. \
this is the 2nd string."
```

10.6.2 Raw String

Raw string means ignoring escape sequences within the string. Just prepend the string with `r` or `R`.

```
r"Newlines are indicated by \n"
```

Listing 10.3: Raw String

When dealing with *regular expression*, we'd better use raw string, which otherwise would require a log of backslash escaping.

Chapter 11

Bash

11.1 Stop List

- The NUL byte is an ASCII *control character* `0x00` (binary `00000000`) resembling `\t \b`. It is a valid character occupying one byte in memory but not visible `printf` can produce them with `\0` in the format spec. GNU/BSD `find` can terminate filenames with them (`-print0`). Bash's `read` can stop (delimit) on them with `-d ''`.
- Bash Manual
- Definitions
- shellcheck
- Expansion $: variable, paramter, command, pathname, arithmetic, history, brace
- Substitution: process, command
- Bash Parser
- Quoting: escape character \, single quotes, double quotes, ANSI-C Quoting
- declare -p name
- `sed -n l`, `cat -e`
- Term *newline* refers to a *visual* and *electronical* new line, which is what is shown (a real new line) when Enter key is pressed.
- Apart from builtins and functions, other (exernal) commands (i.e. `find; awk` are executed in a sub-shell. *pipeline* and process substitution also runs in a sub-shell. You see that, the shell script and commands within it run at different levels.

- A sub-shell is an *enhanced* sub-process, almost an identical copy of the parent shell process. They share the same variables, functions, *export*, and even `$$` equals to that of parent process. In other words, sub-shell inherits almost everything! From Bash 4.0 onward, the BASHPID is set the child process instead. However, variable assignments within sub-shell would not bring side effects to the parent process like `var=1; (var=2; echo $var) ; echo $var`. Read more at FAQ disappear.
- Use builtin `PWD` variable instead of `pwd` command.
- Command-line Tools can be 235x Faster than your Hadoop Cluster
- Command *option* and *argument* are slightly different. Options are usually specified with a hyphen and a single character (i.e. `grep -E`) that is defined by the command author. Arguments are generated by users like `printf "hello , world"`.
- *filename* and *pathname* are used interchangeably.
- Some synonyms for globbing/glob (depending on the context in which it appears) are pattern matching, pattern expansion, filename expansion, wildcard and so on. Unquoted glob does filename expansion. Bash uses glob while `awk`, `sed`, and `grep` use regular expression. Specially, for `find`, strings passed to the `-name` option are used as glob.
- Regular expression; Extended regular expression. `!re !ere !bre`
- Pattern matching; Extended pattern matching `shopt -s extglob`. `!pe`
- `echo "\n" ; printf "\n" ; printf '%s' $'\n'`. A literal backslash followed by a literal `n` (`\n`) within a double or single quoted string preserve their literal character meaning insteat of *newline*. To print as a newline, use `printf`. Whether `\n` is treated as a newline depends on the context. To insert a lieteral newline, use ANSI-C Quoting `$'foo\nbar'`.
- set vs. shopt.
- In Bash man page, check `Lists` for how Bash commands

11.2 Bash Parser

Bash Parser gives details on how Bash processes script files or command lines, which helps we understand the basic logic behind. Figure 11.1 is a simplified image illustration.

Figure 11.2 is a better presentation. From which, we find *expansion* plays a critical role in the whole procedure.

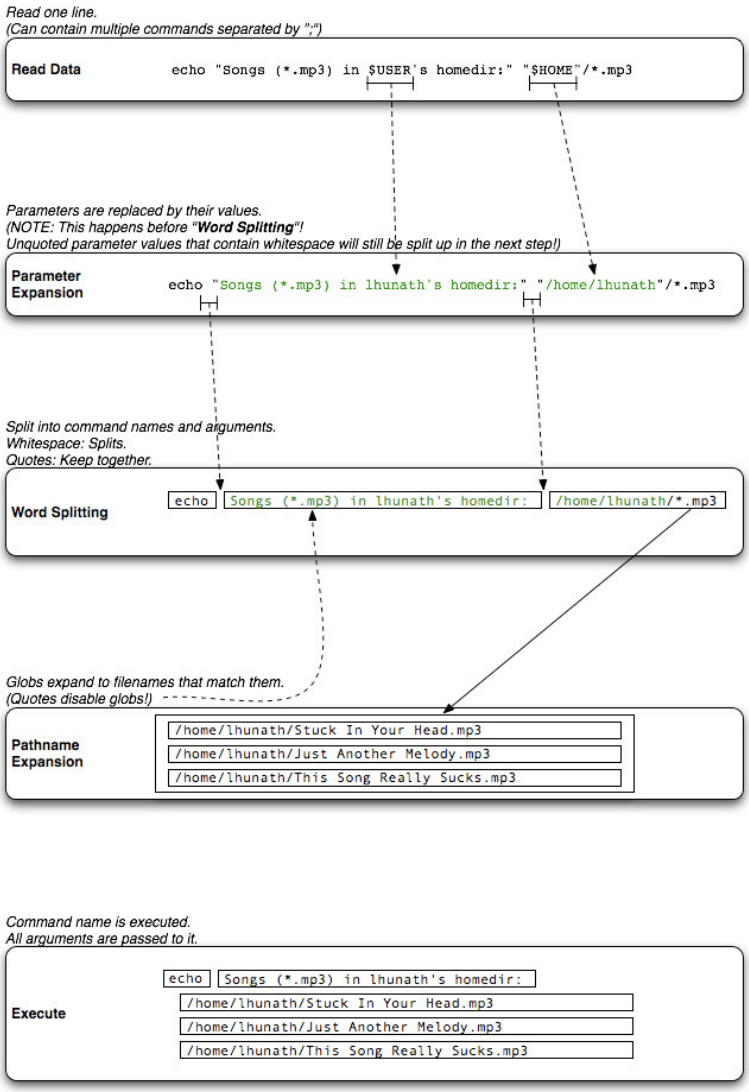


Figure 11.1: Bash Parser

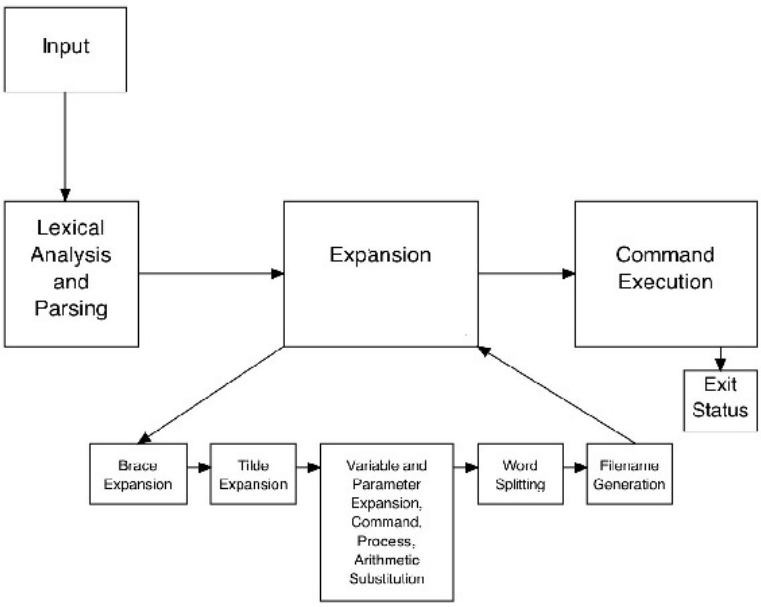


Figure 11.2: Bash Architecture

Basically, the parser carries out the following procedures:

1. Read script source line by line. For each line,
2. Process quotes.
3. Split a line into commands. For each command,
4. Process redirections and brace expansions.
5. Perform expansions, including parameter expansion, comamnd/process/arithmetic substitution etc.
6. Word Splitting: split the command into command name and arguments list.
7. Execute the command.

11.3 CRLF

This sections talks about keys (and their names) and keybord.

We have *physical* keyboard with labels like 1, 2, Enter, Delete etc. The physically pressed keys are captured by OS and then translated *integer keycodes*. Programs read the keycodes and interpret them as appropriate *logical* keys.

We can change keycodes passed from OS to a program like switching left Ctrl and Caps. When Caps key is pressed, the Ctrl keycode is sent to the program.

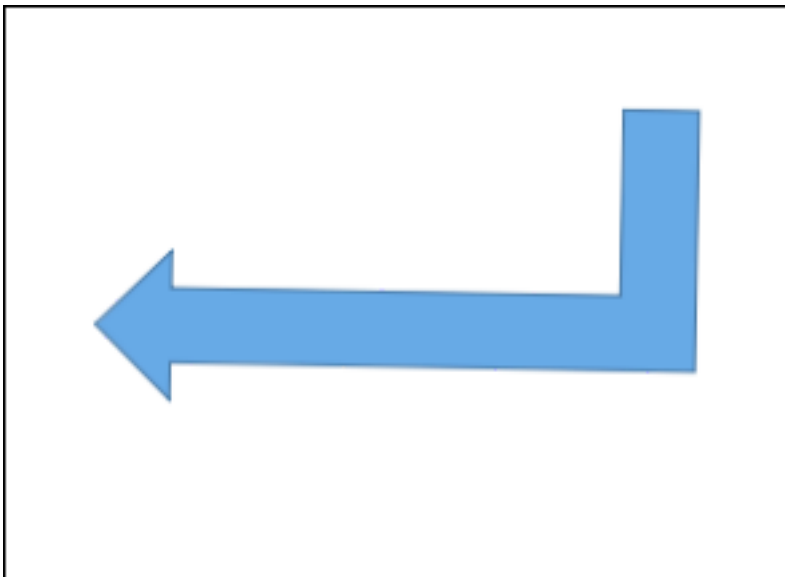


Figure 11.3: Enter/Return Key

Several logical keys have their own names. Specifically, DEL, ESC, LFD, SPC, RET, TAB, and *newline* all stand for themselves when seen in online posts. Name `C-m` (pressing the Return key) is translated to logical RET. `C-k` for pressing Control and `k` simutaneously. `C-j` is translated to LFD. Don't be fooled. Logical name `DEL` corresponds to `Backspace` on keyboard, not Delete.

Then let's have a look at ancient typewriter, comprising *Carriage Return* that helps feed a new line. When a new line is required, the carriage return pulls back the typing head and then a new paper line is fed up. So in the old times, a new line consists of a Carriage Return and a Line Feed.

In the PC age, the corresponding keyboard key is Enter. When it is pressed, the logical name is ENTER, also called (*electronical*) *newline*. So it is not uncommon that the Enter key has an arraw line as depicted in 11.3. The vertical line means line feed while the horizontal arrow means carriage return.

The two operations can be named as `CR` and `LF` respectivelly. Programing languages, Bash included, use `\r` and `\n` to denote the two names. In Linux, it is `\n`, while in Windows it is `\r\n`. Mac OS, to the contrary, is `\r`. We call this *line ending scheme*. It is critical to be aware of line ending difference, especially when web techniques are involved like 11.1.

- `C-m, ^M, \r, Carriage Return, RET` are all the same thing.
- `C-j, ^J, \n, Line Feed, LFD` as well.

11.4 Code Disassembling

I think it is a good practice to retrospect what have been learned recently, reviewing progress and summarize experience. Additionally, I'd like to make use of such chances to format the knowledge in my own wording. Such an *output* process testifies whether knowledge settles down and consolidates.

This section disassembles the `missTime` code line by line. Especially, I should explain *how* and *why*. The complete code is attached at A.4.

11.5 Quoting

Quoting is used to remove the special meaning of certain characters (i.e. whitespace for word splitting) or words to the shell. For example, it prevents reserved words from being recognized as such. It can also prevent parameter expansion.

There exist multiple quoting mechanism, namely:

- the escape character \
- single quote
- double quote
- ANSI-C quote

A non-quoted \ preserves the literal meaning of the next character, with the exception of newline. `\newline` is treated as a line continuation like

```
echo \
'hello, world'

echo 'hello,
world'
```

For quotes, there is no need to append the trailing backslash. The newline splits the string into two lines.

Single quotes preserve the literal meaning of each character within the quotes, except single quote (even preceding by a backslash which also preserves its literal meaning) like:

```
echo 'what\'s your name?'
```

The backslash does not serves as escape. This is an illegal Bash statement and it actually is parsed as three parts, namely `<what\>`, `<s your name?>` and the final apostrophe `'`. The second part is not quoted and the third single quote is unbalanced and Bash expects another single quote. The revised versions are legal but may not be our desired results.

```
echo 'what\'s your name?
echo 'what\'s your name?'jim'
```

But if we `printf '\n'`, a newline is correctly printed. Why? That is because `\n` is literally passed the the command `printf`. It is the responsibility of `printf` to interpret `\n` as newline. For Bash, they are just a backslash `\` and character `n`, not a newline.

Apparently, single quotes is kind of *strong* quoting method to preserve character meaning. But we can precede it by backslash $ to preserve the escaping ability of \, where escaped characters are replaced by their ANSI-C Quoting. For example, `$'\t\''` presents a horizontal tab and a single quote itself. Therefore, if we want to include a literal single quote within singles qoutes, we can precede the quotes with dollar `$`.

Double qoutes is somewhat relatively flexible. It preserves the literal meaning of all characters, except that of `$`, `` ` ``, `\` etc. Within double quotes, `\` can be used to escape `$`, `"`, `` ` ``, `\` and `newline`. For example,

```
echo "
    looooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooong
    \
,line. \$"

echo '
    looooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooong
    \
,line. \$'
```

A double-quoted string preceded by a dollar sign `$"string"` will cause the string to be translated according to the current *locale*. If the current locale is C or POSIX, the dollar sign is ignored. If the string is translated and replaced, the replacement is double-quoted. Attention, this is not parameter expansion, nor ANSI-C quoting.

11.6 printf and echo

```
printf 'Usage: ./missTime <number-of-test> <input-file>\n\n'
```

Use Bash's built-in or system's `printf` instead of `echo`. `printf` supports all C language format to allow flexible output. For instance, We can use it to assign variables by `-v` option:

```
printf -v var "hello, world\n"; declare -p var
```

To check Bash's `printf`, just run `help printf`. My Gentoo's system version is located at `/usr/bin/printf` that is part of GNU coreutils. In terminal emulator, `type -a printf` lists both versions. Unlike `echo`, `printf` does not append a *newline* by default.

For the full contention, check Why is printf better than echo? and echo protability considerations.

As the page writes, *never use options/arguments to echo* like `-e`. Also, do not use it to print *uncontrolled data* such as external input (filename, arguments etc. from user).

In summary, `printf` is far more reliable and safer. It litterally output as the format specifies.

A final note, to output sequential dashes like `---`, `printf` can do as follows:

```
printf -- '---\n'
printf '%s\n' "---"
printf '---\n' # error
```

For the principle behind, just read Bash manual, the OPTIONS section. Basically, everything after two consecutive dashes is treated as arguments (i.e. filenames). In other words, it signals the end of command options and disable further option processing.

Command `tee -a` writes to both standard output and files, which can be combined with `printf` like:

```
printf '%s\n' "hello, world" | tee -a file.txt
```

Single quotes are preferred in the format part unless you want parameter expansion there.

Let's have a look at another sample:

```
printf '%s\n' '1d' w | ed -s "$_full"

sed -i -e '1d' "$_full"
```

You will find there is only one format specification `%s` but two arguments `'1d'` and `w` (not quoted). In such case, the format specifiers are reused:

> The format is re-used as necessary to consume all of the arguments. If there are fewer arguments than the format requires, extra format specifications behave as if a zero value or null string, as appropriate, had been supplied.

11.7 Exit Codes and Boolean

```
[[ "$1" = @(-h|--help) ]] && exit 0
```

`exit` terminates the current shell *process* while `return` just terminates a *function* execution.

Both give numeric `0` for success execution and other specific codes for different error types. Here is a list of Bash Reserved Exit Codes.

The relation between Shell exit codes and boolean is as follows: *exit and return codes are just numeric integers*. They are not boolean values. To be logically consistent, we should correctly interpret the integer codes.

Bash treats 0 for true while others (1 included) for false. Numeric 1 is just one instance among the whole set of failure codes. It is not what we might think that false is bound to 1. Let's have a closer look at the procedure.

A sub-shell decides the status of execution. If a desired outcome is captured, it is logically true/successful. Otherwise, different failure outcomes are treated as false.

Unpon successful execution, numeric 0 is returned back to the parent process. Actually, any number could be returned to represent successful execution, but as the link points out 0 is *platform and encoding independent*.

Now that numeric 0 is returned for success and other numerics for failure, the caller or the topmost Bash process should correctly interpret numeric codes to boolean true/false when `if`, `[[ ]]`, or `while` are involved.

Attention that, *true* and *false* are not reserved words in Bash. But we have commands `type -a true false`. Up to now, you may find, many commands has two versions: one from GNU coreutils and the other from Bash builtins.

To test if a command is successfully executed, just test the command:

```
if command ; then : ; fi
```

The next code also works, but it is *stupid*. It runs a new command `[[` to test another command's exit code.

```
# -or-
command; if [[ $? = 0 ]]; then : ; fi
```

Here, we find another command pair `type -a[`. `[[` is only available to Bash.

11.8 read

```
read -r _ _ _ _ip _ < <(ip -4 -o addr show scope global dev bond0)
 _ip="${_ip%/*}"
```

`read` is another Bash builtin that read a line from standard input and split it into fields based on `IFS` [1] . Check details at Word Splitting. You are recommended to include the `-r` argument as it disables bachslash escape.

`read` assigns the first splitted word to the first name, the second word to the second name and so on, with any leftover words assigned to the last name.

Why are there *underscore*s before and after `_ip`? Mainly, they are just placeholders to skip unwanted splitted words, explained in the next section.

Thus, the first three undersocres capture the three words ahead while the last underscore captures everything else, including the IFS characters and leaving the remaining part untouched.

Attention that, `read` uses `IFS` as *field* separator while use `-d` as *record* separator.

[1] The default Internal Field Separator is <space>, <tabular>, and <newline>, namely `$' \t\n'`.

11.8.1 readonly

The builtin `readonly` command with the form

```
readonly [-aAf] [name[=value] ...]

readonly -p
```

marks each *name* as *readonly*. if a *value* is supplied, do assignment before marking as read-only.

11.9 Special Parameters

These parameters may *only be referenced*; assignment to them is not allowed. Let's see how them are expanded.

- `*`: `$*` expands to positional parameter. Without double quotes, each positional parameter expands to a separate word. Within double quotes, all positional parameters expand to a single word with the value of each positional parameter separated by the first character of variable `IFS`.
- `@`: `$@` is similar to `$*`, except that when in double quotes, each positional parameters also expand a separate word.
- `#`: `$#` expands to the number of positional parameters in decimal.
- `?`: `$?` expands to the exit status of the most recently executed *foreground* pipeline.
- `-`: `$-` expands to the current *option* flags as specified upon invocation, by the `set` builtin command, or those set by the shell itself (such as `bash -i`).
- `$`: `$$` expands the process ID of the *topmost* shell. In a sub-shell (i.e. `( )`, it expands the process ID of the invoking shell (upper level) as it is *inherited*. In a sub-shell, use `BASHPID` instead, which expands to the process ID of *current* shell. `echo $$ $BASHPID; ls -l /proc/self; (echo $$ $BASHPID; ls -l /proc/self)`. Also check /proc/self.
- `!`: `$!` expands to the process ID the *job* most recently placed into the background.
- `0`: `$0` expands to the name of shell or shell script.
- `_`: `$_` expands to the last argument of privous simple command. Read the next section.

11.10 Underscore

Firstly, Bash regards *underscore* _ as special parameter. Meanwhile, it is a standalone legal *name* (vairable) and can be part of a variable as well. In short, it is the only *parameter* that is also a valid *variable name*.

```
echo 'hello, world'
declare -p _ ; declare -p _
```

As a special parameter, the value of undersocre _ is assigned *automatically* by Bash process as depicted in the next section, namely *expansion* 11.13. At the very beginning, it is set to the absolute *pathname* of the shell or script file. Subsequently, *expand*s to the last argument of previous simple command.

Though it is a variable, we can **not** assign value to it explicitly. In other words, assignment to it is legal but in vain.

```
_="a"
declare -p _
```

That is the reason we use it as a placeholder for `read` command. Its value is *unstable* and got *overriden* immediately after each Bash command. Values assigned to it take no effects. We can verify this behavior by:

```
read -r _ var <<< 'hello, world'
declare -p _
# -or-
printf '%s\n' "$_"
```

The code outputs `declare -- _="var"` since the last argument of `read` is a name (`var` here).

Pay attention to the difference between *declare* (only the name is required) and *printf* (name should be expanded by `$`).

11.11 extglob

Here are a few *extglob* samples. To turn on this Bash feature, just `shopt -s extglob` *on a newline*.

`extglob` changes the way certain characters are parsed. It is necessary to have a newline (**not** just a semicolon) between `shopt -s extglob` and any subsequent commands to use it. This is because the command line is parsed before `shopt` command is evaluated. Check the parser section before.

Likewise, you cannot put `shopt -s extglob` inside a statement block that uses extended globs, because the block as a whole must be parsed when it's defined; the `shopt` command won't take effect until the block is evaluated, at which point it's too late. In fact as Bash parses the entire statement block before evaluating any of it, you need to set *extglob* outside of the outermost block.

The first example below, does not turn on `extglob` correctly and report error.

```
shopt -u extglob
shopt -s extglob ; touch foo bar ; echo @(foo|bar)
# -bash: syntax error near unexpected token `('

_ip=127.0.0.1 _std_array[6]="https://www.baidu.com/a/b/c/d?v=k"
printf '%s\n' "${_std_array[6]}"
shopt -q extglob; _extglob_set=$?
(( _extglob_set )) && shopt -s extglob
_delete_url=${_reply_array[X-True-Cache-Key]#http?(s)://}
(( _extglob_set )) && shopt -u extglob

bash -c $'shopt -u extglob\nshopt -s extglob ;
url="https://www.google.com/a/b/c" ;
_delete_url=${url#http?(s)://} ;
shopt -u extglob; printf "%s\n" "${_delete_url}"'

shopt -u extglob
var='--help' ; [[ "$var" = @(-h|--help) ]] && echo 'yes'
```

The second case replaces the `://host/` with `://ip/`, which combines the features of *extglob* and *parameter expansion* in the form `${parameter/pattern/string}`. In the pattern part, we should escape the backslash \while in the string part, leave as it is since this part is leterally substituted.

The next two cases show both `${parameter/pattern/string}` and `[[` use `extglob` internally even we turn it off.

But to be consistent and robust, we are still recommended to put it on separate lines. This rule applies to other shell options as well.

11.12 Process Substitution

Still the same code snippet at previous section, we have

```
< <(ip -4 -o addr show scope global dev bond0)
```

`>(list)` or `<(list)` is called Process Substitution that allows a process's input or output to be referred to using a filename, which the process get inputs from or output results to a file. Attention, a sub-shell is spawned when process substitution happens (there is a pair of parentheses there).

The keyword *list* here may be a normal command or a pipeline of commands. No space may appear between the < or > and the left parenthesis), otherwise the form would be interpreted as a redirection. Process Substitution is supported on systems that support named pipes (FIFOs) or the `/dev/fd/` method of naming opened files.

We can just think of the two forms as just filenames, which is done by giving the list a name in the filesystem:

process substitution = filename

For the input form `>(list)`, writing to the file will provide input for the enclosed *list* like:

```
command ... >(list) ...

echo >(true)

cat file.txt > >(wc -l)
```

The first example prints the input filename of the command substitution. Instead, the second *redirects* the output of `cat` to the input filename.

If the output form `<(list)` is used, the file can be read to obtain the output of the *list* like:

```
command ... <(list) ...

echo <(true) # print the output filename
cat <(list)  # print the contents of the output filename

# compare the two output filenames
comm -3 <(sort a.txt | uniq) <(sort b.txt | uniq)
```

Back to the beginning code, the leftmost < is a normal I/O redirection, meaning read from the process substitution. Pay attention to the extra space between the two <, otherwise it would be a Here Document.

11.13 Parameter Expansion

```
1 _num_of_tests=${1:-10}
```

The purpose of this code is to assign value to `_num_of_tests` from either user input or the default `10`.

The special $ symbol of Bash introduces *parameter expansion*, *command substitution*, or *arithmetic expansion*. The terminology - *parameter* - is defined in the manul, please have a read. To put it simple, parameter is an entity that stores values, it can be a *name*, a *number*, or special parameters like `* @ # _` etc., of the list, undersocre is explained in a prior section.

name is what we usually call as *varaible* or *function name*. Also referred to as an *identifier*.

The parameter symbol to be expanded may be enclosed in braces which are *optional* but *safer* as braces separate the variable from characters immediately following it.

The basic form is `${parameter}`. Our case is `${parameter:-word}` that means the expansion of *word* is substituted if *parameter* is null or unset, otherwise the value of *parameter* is substituted.

Here are three extra examples:

```
for f in full real; do declare "_$f"="${_log_time}-${_num_of_tests}-$f.
    log"; done
2 for f in "${_full}" "${_real}"; do echo -n >| "$f"; done
```

This code uses `echo -n` to create an empty new file, which can also be accomplished by `touch` or `printf ''`

11.14 Brace Expansion

Brace expansion is used to generate arbitrary strings. It takes the following forms:

```
# {string1,string2,...,stringN}
2 {a,b,c,d}
# {<START>..<END>}
4 {1..10}
# {<START>..<END>..<INCR>} (Bash 4)
6 {1..10..2}
# <PREAMBLE>{........}
```

```
# {........}<POSTSCRIPT>
# <PREAMBLE>{........}<POSTSCRIPT>
a{b,c,d,}.txt
```

Both the PREAMBLE and POSTSCRIPT parts are optional. The last example above can expand to `a.txt` for the trailing comma `,`.

Within the braces, we can place either a series of comma-separated strings, or a *sequence expression*. The should be *no* space between the list elements. Results of expanded strings are not sorted; let to right order is preserved.

A sequence expression takes the form `{x..y[..incr]}`, where `x` and `y` (both inclusive) are either intergers or single characters, and `incr`, an optional argument, is an integer. Both `x` and `y` must be of the same type.

Supplied integers may be prefixed with 0 to force each term to have the same width like `for i in {01..12}; do echo $i; done`.

It is different to globbing that matches filenames by default (unless used in parameter expansion).

> This mechanism is similar to pathname expansion, but the filenames generated **need not exist**.

That means `{foo,bar}.txt` expands to `foo.txt` and `bar.txt`. Whether the two strings are used as filenames or not depends on how you use it.

```
for f in {foo,bar}.txt ; do echo "$f" ; ls "$f" ; done
```

But glob `*.txt` only expands to *existing* files. If no files are matched, it expands to the glob itself unless `nullgob` is enabled which makes it expands to *null string*.

```
bash -c $'shopt -u nullglob\nfor f in *.txt; do ls "$f"; done'

bash -c $'shopt -s nullglob\nfor f in *.txt; do ls "$f"; done'
```

From section Bash Parser 11.2, we know brace expansion happens before all the rest expansions. Any characters special to other expansions are preserved in the result: *strictly textual*. So brace expansion makes

```
echo {a,b}$PATH
```

be

```
echo a$PATH b$PATH
```

before PATH is parameter-expanded.

Similarly, a brace expansion with range like {1..200} can not be expressed like:

```
a=1 b=200
echo {$a..$b}
```

Actually such code does not satisfy a legal barce expansion requirement. Within the *unquoted braces pair*, there should be *at least* a comma or a sequence expression.The textual $a and $b (not parameter-expanded), the dots inbetween included, neither contains a comma nor are sequence expression. Hence, during the brace epxansion phase, the code is ignored. When it comes to the parameter expansion, it becomes a single string {1..200}.

Brace construction is normaly used when the common prefix or suffix of strings to be generated are much longer:

```
mkdir /usr/local/src/bash/{old,new,dist,bugs}
#
chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}
#
echo {{A..Z},{a..z}}
#
echo {A..Z}{0..9}
```

You will find from the code, brace expansion can be nested. It also supports concatenation.

11.15 Parameter Transformation

```
printf 'curl -ksSI '
# 4.4
printf -- '-H %s ' "${_headers_array[@]@Q}"
printf -- '-H %s --resolve %s %s\n\n' "x-c3-debug:enabled" "${
    _resolve@Q}" "${_url@Q}"

# < 4.4
printf -- '-H %q ' "${_headers_array[@]}"
printf -- '-H %q --resolve %q %q\n\n' "x-c3-debug:enabled" "${_resolve}
    " "${_url}"
```

Since Bash 4.4, parameter transformation is supported in the form ${parameter@operator }. Of the operators, Q expands the parameter as a string that is the value of parameter

quoted in a format that can be reused as input like `ar=(a 1 "b ; c") ; printf '%s\n' "${ar[@]@Q}"`.

Before Bash 4.4, we can resort to `printf '%q'` which quotes the argument in a way that can be reused as shell input.

11.16 regex Test

```
_headers_regex=$'\^\~\$headers=\'(.*)\)@\|#\(\''
_headers_regex='\^\~\$headers=\''(.*)\)@\|#\('\'

_https_regex='^https://'
[[ "${_int_log}" =~ $_regex ]] && printf '%s\n' "${BASH_REMATCH[1]}"
```

Bash support the extended regular expression by *conditional command* `[[` with binary operator `=~`.

If the pattern is properly matched, results is assigned to built-in variable `BASH_REMATCH`. It is an array variable with index 0 the portion of the string matching the entire regular expression (the complete match, **not** the whole string). The element with index *n* is the portion of the string matching the *n*th parenthesized sub-expression (capture group). If there are nested parenthesis level, the inner index is that of the outer level plus 1.

`BASH_REMATCH` is *read only*. We cannot change its value. Usually, just assign it to a temporary variable.

Pay attention the two methods to define the regular expression. The first one is preceded by a `$` within which `\'` is correctly interpreted. While the 2nd version, put the `\'` outside of single quotes`'foo-bar'`.

11.17 Regular Expression

To learn regular expression, read [Quickstart](https://www.regular-expressions.info/quickstart.html) first. Check the bit-by-bit authoritative tutorial for details.

Pay attention to the terms character class `[ ]` and capture group `( )`. A character class match one and only one character. So `a[1-9]` does not match a single character `a`.

In regular expression, a dot matches any character except visual newline. A negated character class matches a newline instead like `[^a-z]`. To avoid that, use `[^a-z\r\n]`.

Regex is not intended for or good at *inverse* search. But we can mimic this behavior by *lookaround* at Lookahead and Lookbehind Zero-Length Assertions. It is quite useful when we want to exclude something when matching.

For example, if the command is `grep`, use option `-P` to enable PCRE engine. The code below excludes entries of which the URI part does not contain *itunes-assets*.

```
cclog hpc access 201902141200 201902151200 | \
grep -P -m2 'http://aod\.itunes\.apple\.com/(?!itunes-assets).*headers
    ='
```

However, we can resort to `-v` to do the same work.

```
cclog hpc access 201902141200 201902151200 | \
grep -m2 'http://aod\.itunes\.apple\.com/.*headers=' \
grep -v 'itunes-assets'
```

So when to use *lookahead* and when to use *lookbehind*? From my experience, if you don't want to match something *immediately* following a *literal string*, then use *lookahead*. Similarly, if you don't want to match something immediately preceding a literal literal string, use *lookbehind*.

Like `^` and `$` (for start and end of *string*), `\b` is also an anchor for start and end of *word* like `\bhello\b`. It can be called as *boundary*. The counterpart `\B` matches at every position where `\b` cannot match.

To match a valid IPv4 address, use:

```
^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$
```

11.18 Array

Bash provides one-dimentional *indexed* and *associative* array variables. The *declare* builtin can explicitly declare an array:

```
declare -a indexed_array # optional
declare -A associative_array=()
```

Indexed array can be any vairable, and hence a declaration is optional.

> Any reference to a variable using a valid subscript is legal, and bash will create an array if necessary.

On the contrary, associative array *must* be declared before usage.

Indexed arrays are zero-based while associative array resembles Python *dictionary*, with key and value pairs stored. Pairs are not sorted.

When assigning elements to an array, please enfold the *subscript* with square brackets.

```
indexed_array[0]=12
indexed_array=( [1]='a' 'b' 'c' )
printf '%s\n' "${indexed_array[@]}"
#
declare -A associative_array=( ['addr']='usa' )
associative_array['name']='jim'
associative_array=( ['age']=30 )
for key in "${!associative_array[@]}"; do printf '%s: %s\n' "$key" \
"${associative_array[$key]}"; done
```

Array elements can be referrenced by the form `${name[subscript]}`. The curly braces are required to avoid conflicts with filename expansion operators.

> `${`. Without curly braces parameter expansions refer to the *longest valid variable name* or *shortest positional parameter*. `"${var}bar"` expands the parameter named *var* while `"$varbar"` expands *varbar*. `"$123"` references `argv[1]` and `"${123}"` references `argv[123]`. Braces are requried for positional parameters > 9, special PEs, and array expansions: `${10}`, `${var##pat}`, `${arr[5]}`. BRACES AREN'T A SUBSTITUTE FOR QUOTES!

If the *subscript* is omitted, `$name` implies a subscript of zero `0`, which suggests *name* is an indexed array or an associative array with a key of zero `0`.

```
unset ar; declare -A ar=( [0]=a ["hello"]=1 )
declare -p ar; printf '%s\n' "$ar"
[[ -v ar ]] && echo "yes"
```

To print array:

```
# indexed array
printf '%s\n' "${idexed_array[@]}"
# associative array
for key in "${!idexed_array[@]}"
do
    printf '%s: %s\n' "$key" "${idexed_array[$key]}"
done
```

The code above uses special subscript `@` that expands to positional parameter. Another special subscript is `*`, when double quoted, expanding to a single word with each element separated by the first character of the `IFS` which by default is *space*. Hence:

1. For printing, use `@` instead of `*` unless explicitly required.
2. When setting custom `IFS`, pay attention to the character sequence.

Now, let's have a look at the following code:

```
declare -A _reply_array=()
while IFS=' :' read -r key value
do
    [[ -n "$key" && "$key" != [[:space:]] && "$key" != + ]] || continue
    _reply_array[${key,,}]=${value%$'\r'}
done < "curl-reply.log"
```

Listing 11.1: curl CRLF

1. Associative array must be delcared.
2. The first character of `IFS` is set to space.
3. `-r` argument of `read` is adopted.
4. `continue` starts the next loop instantly.
5. *key* is transferred to lower case by parameter expansion in the form `${parameter,,pattern}`. *pattern* is omitted.
6. Remove trailing *carriage return* `\r` from `curl`.

The sixth item is really hostile to programmers. Basically, `curl` feeds back messages with CRLF instead of `\n`. We can check this by `sed -n l`. To reuse some returned headers, we should firstly remove the extra `\r`.

11.19 Command Substitution

Similar to Process Substitution 11.12 above, command substitution performs the expansion by executing the command within a sub-shell.

It allows the output of a command to substitute the command itself, with trailing newlines delete. We can just think of command substitution as a string and using the output of a command as an argument. No filenames are involved.

It takes the following two forms. The *backquote* form is deprecated and discouraged.

```
$(command)

`command` # deprecated
```

If the substitution appears within double quotes like `"$(command)"`, *word splitting* and *filename expansion* are not performed on the results, which is a *preferred* method.

Command substitution is useful when we want to assign a command output to a variable like `_date="$(date -Idate)"`.

The command substitution `$(cat file)` can be replaced by the equivalent but faster `$(< file)` (without sub-shell). The later one is a *specical* case of command substitution. It does not invoke any command and Bash just substitutes it with the contents of file.

11.20 Assignment and Simple Command

```
IFS=$'\n' _log_array=( $( awk -F'[[:space:]]*\\)@(in|out)#\\([[:space:]]*' \
                            '{ for (i = 1; i <= NF; ++i) print $i; }' <<< "${line}" ) )
# better
mapfile -t _log_array < <( awk -F'[[:space:]]*\\)@(in|out)#\\([[:space:]]*' \
                            '{ for (i = 1; i <= NF; ++i) print $i; }' <<< "${line}" )
#
IFS=';' read -ra _headers_array <<< "${BASH_REMATCH[1]}"
_headers_array=( "${_headers_array[@]/#/-H}" )
```

In the first assignment uses `awk` command substitution with output assigned to an indexed array directly. This substitution is not double-quoted, so word splitting and pathname expansion happens before assignment. The second assignment uses `awk` process substitution and mapfile 11.20.7 (explained in a standalone section), which is much *faster* and *safer* method.

Why, in this section, I want to talk about assignment and simple command? We can think of simple command as just a command name followed by its arguments. From Simple Command Expansion, we find:

> If no command name results, the variable assignments affect the current shell environment. Otherwise, the variables are added to the environment of the executed command and do not affect the current shell environment. If any of the assignments attempts to assign a value to a readonly variable, an error occurs, and the command exits with a non-zero status.

The only command awk is executed in a sub-shell, The two assignments IFS=$'\n' _log_array= affect the current shell. That is to say, from here on, the new IFS value is changed for the whole Bash process. Instead, the IFS associated with the read statement only affects the read command itself.

Let's reformat the code like:

```
IFS=$'\n'
_log_array=( $( awk -F'[[:space:]]*\\)@(in|out)#\\([[:space:]]*' \
                    '{ for (i = 1; i <= NF; ++i) print $i; }' <<< "${line}" ) )
IFS=$' \t\n'
```

It becomes more understandable and more clearer.

Extra notes :

Recall that indexed array can be declared by -a argument. read command supports reading into an indexed array by the same argument.

"${_headers_array[@]/#/-H}" is another example of parameter expansion in the form ${parameter/pattern/string}. It inserts -H in front of each array element.

11.21 Commands

11.21.1 sed

sed, as described in the manual, is a stream editor for filtering and transforming text.

We can use it to delete empty lines:

```
sed -i.bak '/^[[:space:]]*$/d' file.txt
sed -i.bak '/[^[:space:]]/!d' file.txt
```

The -i[SUFFIX] or --in-place[=SUFFIX] option edits files in place (makes backup if SUFFIX supplied).

Form /regexp/ matches lines against the regular expression. In the example above, ^[[:space:]]*$ matches empty lines. The traling command d means to delete matched lines.

The variant version is somewhat not so obvious but usually faster if the file has less empty lines. Pay attention to the exclamation mark `!` before command `d`. This version means as long as a non-space character is matched, don't delete the line.

Another useful form is `s/regexp/replacement/` that matches file lines agaist the *regex* and *replace*s the matched parts with *replacement*. We can use `&` in the *replacement* to represents the matched parts like `s/hello/=&=/` means to enclose string *hello* with symbol `=`.

The snippet below replaces *google* with *bing* if the URL contains *foo*. Note that in the regular expression part, dot should be escaped while in the replacement part *not*. A version with `awk` are also provided for reference. Please read more in `awk` section.

```
var=$'https://www.google.com/foo?v=xyz\nhttp://www.baidu.com/bar?v=uvw\
    n'
printf '%s\n' "$var"
sed '/foo/s/www\.google\.com/www.bing.com/' <<< "$var"

var=$'https://www.google.com/foo?v=xyz\nhttp://www.duckduckgo.com/bar?v
    =uvw\n'
printf '%s\n' "$var"
awk -F'/' 'BEGIN { ORS="/" } /foo/{ $3="www.bing.com" ; for ( i=1; i<=
    NF; i++ ) print $i }' <<< "$var"
```

In the meanwhile, special escapes `\1` through `\9` refers to the corresponding matched sub-expressions in the *regexp*. For example, `s/(h.)l(.o)/\2/` will replace the word *hello* with *lo*.

To match a continuous lines block, we have:

```
sed -n '/addr1/,/addr2/p'
#
printf 'hello\nworld\nbash\nhello\nchina\n' | sed -n '/he/,/wo/p'
```

- the line which addr1 matched will always be accepted, even if addr2 selects an earlier line.
- If addr2 is a *regexp*, it will not be tested against the line that addr1 matched. In other words, if it is other forms (check man page, i.e. number), test is required.
- If addr2 is not matched, all lines starting from addr1 to the end of file are matched.

Similarly, `awk` has also such block address feature. Check the relevant section below.

Here is an sophiscated example:

```
~$ cat file
# Title: foobar
# Subject: Subject
# Body: Message body

# Title: foobaz
# Subject: Another Subject
# Body: Another message body

sed '/Title/,/Subject/s/foo/test/' file
```

Finally, to differentiatethe concepts of *pattern space* and *hold space* is important to master `sed`.

11.21.2 ed

`ed` is line-oriented text editor. `Sed` is a stream editor.

A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits *scripted* edits (such as `ed`), `sed` works by making **only one pass** over the input(s), and is consequently more efficient. It is `sed`'s ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

Due to the *stream* and *one pass* features of `sed`, it cannot move back to earlier lines, where `ed` comes into usage.

```
ed -s "${_full}" <<< $'$-3d\nw'
ed -s "${_full}" <<< $'$-3d\n,p'
```

`$` refers to the last line and `$-3` means the last but 3rd line. Similar to `sed`, command `d` means to delete the matched line.

Command `w` means save the file. Usually `ed` allows only one command in a line except `p` etc. Consequently, we need `\n` to separate `d` and `w`. `sed` on the other hand, allows semicolon `;` to separate different commands.

11.21.3 awk

From the manual, `awk` is a pattern scanning and processing *language*. You see! It is a programming language, not just a command line tool.

There exist multiple awk implementations, of which gawk is, by default, deployed on all Linux distributions. This version is the standard implementation for Linux, while the original awk was written for Unix v7. The original awk authors then released a new version called nawk or bawk but rarely used. Another version is mawk that runs faster as it is based on a byte-code interpreter. On Linux systems, awk is a symbolic link to either gawk or mawk. To speed up awk, we can set LC_ALL=C awk 'foobar'.

The basic form is as:

```
gawk [ POSIX or GNU style options ] -f program-file [ -- ] file ...
gawk [ POSIX or GNU style options ] [ -- ] program-text file ...
```

awk reads source code either from a external file or from inline text.

We usually write *program-text* like pattern { action statements }. Action statements (enclosed in braces, { and }) will be performed on the matched lines. Either the *pattern* or *action statements* part may be missing, but *not* both. A missing action is equivalent to { print }.

pattern can be but not limited to */regular expression/*. The surrounding forwardslash is required like awk '/[a-z][1-9]/ {print $1}'. If the forwardslash is omitted, awk will treat it as a variable which usually evaluates to null as it is probably undefined.

Another pattern is *Relational expression* with operators like + - * / % etc. It is usually used to test whether certain fields match certain regular expressions with ~ or !~ like awk '$4 ~ /^https/ {print $4}'. Also, surround the regular epxression with forwardslash.

However, a literal string (i.e. http://) with qoutes is acceptable like awk '$4 ~ "^https:\\/\\/" {print $4}'. But we'd better use double backslash \ to esecape special characters.

awk also has BEGIN and END patterns which are not tested against the input and can have their own action statements. Actions defined in the BEGIN are performed before any input read while those of END are performed when input is completely read.

We usually can change built-in variables in BEGIN part like FS RS OFS ORS, where F refers to *field* while R means *record*. Prefix O for *print output*.

```
awk 'BEGIN { print "Count \"hello\"" }
/hello/ { ++n }
END {print "\"hello\" appears in", n, "lines." }' file.txt

var=$'https://www.google.com/foo?v=xyz\nhttp://www.duckduckgo.com/bar?v
    =uvw\n'
```

```
awk 'BEGIN { FS=OFS="/" } $4~/^foo.*/ { $3="s3.bing.net" ; print }' <<<
     "$var"
```

For more details, refer to the PATTERNS AND ACTIONS section of the manual.

Here is a real case:

```
awk 'BEGIN {IGNORECASE=1}; /[-_]cache:/ { print substr($0, 3) }'
```

Like sed, awk also accepts range pattern `pattern1, pattern2`. It matches all input records starting with a record that matches pattern1, and continuing until a record that matches pattern2, inclusive. It does not combine with any other sort of pattern expression.

```
awk '/pat1/,/pat2/'
#
printf 'hello world\nchina\n' | awk '/he/,/wo/'
printf 'hello world\nchina\n' | sed -n '/he/,/wo/p'
#
printf 'hello\nchina\n' | awk '/he/,/wo/'
```

If a record matches both patterns, then that single record is also regarded as a range, which is slightly different from sed.

Attention that, awk does not support *lookeahd* or *lookahead* since it uses POSIX Extended Regular Expression (ERE).

grepawk

`!grepawk` writes:

```
Awk can do almost everything grep can do. Instead of doing grep 'foo' |
     awk '{ statement }', try awk '/foo/{ statement }'
```

11.21.4 grep

`grep` print lines matching a regular expression pattern. It has has two extensions, namely `egrep` and `fgrep`. `egrep` is equivalent to `grep -E`, while `fgrep` is equivalent to `grep -F`.

Key Binding	Readline Function
C-h	DEL, Backspace
C-_, C-x C-u, C-/	undo
C-l	clear the screen, clear
C-r	backward search
C-s	forward search

Table 11.1: Readline Key Bindings

egrep uses the *extended regular expression* engine. Check the REGULAR EXPRESSIONS section for details. Specially, grep -P uses PCRE engine (that of Perl) which supports *lookaround* as stated in a previous section.

egrep interprets pattern as an extended regular expression while fgrep interprets it as a list of fixed strings separated by *newline*s.

By default, it prints the matched lines. If we suppliy the -v option, not-matched lines are printed.

Another special option is -o that prints only the matched parts instead of the whole line.

11.21.5 readline

GNU Command Line Editing is provided by Readline Library allowing users to edit command lines as they are typed in. Both Emacs and Vi editing modes are available:

```
# enable
set -o emacs
set -o vi
# disable
set +o emacs
set +o vi
```

I am astonished that readline copies the key bindings from Emacs, though it supports that of Vi editor. Table 11.1 lists some common key bindings:

If the terminal happens to turn on Flow Control, then C-s will freeze your terminal. Use C-q to disable it.

11.21.6 xargs

Compare the following two commands, we find that the first line just prints `--help` while the second prints the help message of `cat` command.

```
echo '--help' | cat
# vs
echo '--help' | xargs cat
```

`xargs` builds and executes command from standard input. It mainly reads items from standard input, delimited by blanks and newlines, and pass the input to *command* as arguments.

Actually, the idea is to concatenate a few items together and build a new command to execute. It is sort of converting STDIN contents to the command's arguments, not passing the contents to as input.

In the first example above, `cat` uses STDIN as the argument. And the conents of the argument is literl string `--help`. But in the second example, `cat` uses the contents of STDIN as its argument with the help from `xargs`. From this perspective, we find the necessity of `xargs` in spite of just *pipeline*.

`-d` option changes the default delimiters (blank and newline) like `echo '11@22@33' | xargs -d '@' echo`.

When building the new command, `xargs` may encounter filenames containing newlines or blanks which will broke the results as the filename will be splitted.

This issue could be solved by passing `-0` option to `xargs`, replacing the default delimiters to *null character* `\000`.

Obviously, `-0` option is equivalent to `-d'\0'`. This option is usually combined with `find ... -print0 | xargs -0 ...` . Check the *235x Faster* link in section 11.1 for example.

By default, `find` prints each found file followed by a new line. But `-print0` removes the newline:

```
find . -type f -name *.txt -print | sed -n l
# ./test.log$
# ./sample.log$
find . -type f -name *.txt -print0 | sed -n l
# ./test.log\000./sample.log\000$
```

For safety, `xargs` supports interactive execution by `-p`, prompting the user about whether to run the built command.

Please further read xargs 命令详解，xargs 与管道的区别.

11.21.7 find

`find` is a powerful command, especially when followed `-exec` option. It usually takes the form:

```
find [-H] [-L] [-P] [-D debugopts] [-Olevel] [starting-point...] [
    expression]
```

`-H -L -P` control the treatment of symbolic links. The `-L` option follows symbolic links while `-H` does not follow symbolic links, except while processing the command line arguments.

The term *starting point* means a list of names of files or directories to be examined. By default it is the current directory, namely the single dot.

The *expression* part is complicated. It is a kind of query specification describing how we match files and what we do with the files that were matched. An expression is composed of a sequence of things:

```
<POSITIONAL OPTIONS> <GLOBAL OPTIONS> <TESTS> <ACTIONS> <OPERATORS>
```

Check the relevant sections in the manual, to find out what they are.

Of the list, OPERATORS join together the other items within the expression. They include for example `-o` (meaning logical OR) and `-a` (meaning logical AND). Where an operator is missing, `-a` is assumed.

Here are a few examples of `find`:

```
find -L /path/to/search -type f \( -iname "*filename*" -o -name '*.txt'
     \)

find -L /path/to/search -type f -iname '*.apk' -printf "%f\n" -exec cp
    -fv '{}' /path/to/copy \;

find APPs/ -maxdepth 1 -type f -iname '*.apk' -exec bash -c 'for file;
    do file="${file##*/}"; mkdir -p ROM/system/preset_apps/"${file%.apk
    }"; cp APPs/"${file}" ROM/system/preset_apps/"${file%.*}"; done'
    bash '{}' +
find -P history/ -type f -name '*.HEIC' -exec bash -c 'for file; do mv
    -v "$file" "${file%%.*}.heic"; done' 'bash' {} +
find ROM/system/media/wallpaper/ -depth -type f ! -path '*/wallpaper_15
    .png' -delete
find ROM/system/media/wallpaper/ -depth -type d -empty -delete

find -L . -type f -iname '*abc*' -exec bash -c 'mv "$0" "${0/abc/def}"'
     '{}' \;
export _pkg_name
```

```
12 find -H  -maxdepth 1 -type f \( -name '*.jpg' -o -name '*.png' \) -exec
       bash -c 'for img; do mv "$img" pics/"${_pkg_name}"; done' bash
     '{}' +
   find . -type f -name '*.jpg' -exec bash -c 'for f; do mv $f ${f//"$0"};
       done' $'\302\240' '{}' +
```

1. `-name` and `-iname` uses glob after *word splitting*. It is recommended to quote the glob (explained below).
2. The surrounding parentheses force prededence of operators.
3. `-type`, `-iname`, and `-name` are all TESTS.
4. `-printf` and `-exec` are ACTIONS.
5. `-exec command ;` and `-exec command {} +` are different. The first form executes the *command* once for each file. The string special `{}` is replaced by the current file name being processed. Both `{}` and the semecolon `;` may be quoted (i.e. `'{}'`) or escaped (i.e. `\;`) depending on the SHELL. The symbol `{} +` appends each selected file name at the end and runs the the *command* with all the files as arguments. The number of filenames is only limited by the system's maximum command line length. If the command exceeds this length, the command will be called multiple times. For example `-exec ls {} +`. After the `;` and `+`, other ACTIONS can be supplied.
6. `-maxdepth` and `-mindepth` are GLOBAL OPTIONS.
7. `-exec bash -c 'foo-bar' bash '{}' +`. The second literal string *bash* serves as the `$0` positional parameter to the Bash shell while `'{}' +` serves as others. Without the second *bash* string, the very first file name would be the zeroth positional parameter like `-exec bash -c 'mv "$0" "${0/abc/def}"' '{}' \;`. This code will replace the *abc* part of the zeroth positional parameter with *def*.

From Bash manual, command `for` takes the form:

```
for name [ [ in [ word ... ] ] ; ] do list ; done
```

If the *in word* part is omitted, *name* expands to the positional parameters.

The following two lines only differ whether a explicit `0` positional parameter is supplied. In the first case, `shift` makes `for` to skipt the *bash* string. Otherwise, *bash* would be processed as file name.

```
-exec bash -c 'shift; for f; do foo-bar; done;' bash '{}' \;
2
-exec bash -c 'for f; do foo-bar; done;' '{}' \;
```

```
4
-exec bash -c 'shift; for f; do foo-bar; done;' '{}' \;
```

If `-type d` TEST is supplied, then probably the `0` positional parameter is the current directory, namely the single dot. Sometimes we don't want to it to be processed within `for`. That is where the third case comes into usage. Alternatively, we can use the GLOBAL OPTION `-depth` which processes files within before the directory itself.

Hence, some of the examples above should be rectified!

Now let's have a look at code:

```
set -x
2
touch main.conf ; find . -type f -name main*
4touch main.conf main1 ; find . -type f -name main*
6touch main.conf main1 ; find . -type f -name 'main*'
8set +x
```

Without quoting, `main*` expands to filenames before passed to `find`. The second `find` commands becomes

```
find . -type f -name main.conf main1
```

That is an illegal command form. Hence the second example reports:

> find: paths must precede expression: 'main1' find: possible unquoted pattern after predicate '-name'?

So, always quote the glob after `-name` option.

11.21.8 mapfile

`mapfile` reads lines from the standard input into the indexed array variable *array*, or from file descriptor `FD` if the `-u` option is supplied. The built-in variable `MAPFILE` is the default array is no array argument is supplied. It has a built-in synonym `readarray`.

```
mapfile -t _log_array < <( awk -F'[[:space:]]*\\)@(in|out)#\\([[:space
    :]]*' \
                              '{ for (i = 1; i <= NF; ++i) print $i;
    }' <<< "${line}" )
#
mapfile -t -d ';' _headers_array <<< "$var"
```

The `-t` option removes trailing newline from from each line input. `-d` options can set a new delimiter than the default newline.

Please read section **??** to find out how `while` loop achieves the same goal.

11.21.9 time

Before opening this section. I want to talk about a bit on Command Grouping. Command grouping groups a list of commands to be executed as a unit. It takes two forms:

```
( list )

{ list; }
```

The parentheses executes commands of the *list* in a sub-shell environment. while curly braces executes them in current shell.

- Especially, the trailing semicolon is required.
- Curly braces are *reserved words* of Bash. So they must be separated from the *list* enclosed by by *blank*s or other shell metacharacters.
- Parentheses are *operators*, and are recognized as separate tokens even they are not separated from the *list*.

When commands are grouped, redirections may be applied to the entire command list. For example, the output of all the commands in the list may be redirected to a single stream. This is how we capture the *time* output.

```
{ time curl -kLsSvo /dev/null "${_headers_array[@]}" \
            -H "x-c3-debug:enabled" --resolve "${_resolve}" "${_url}"
    2>&1 | \
       awk 'BEGIN {IGNORECASE=1}; /<[[:space:]].*[-_]cache:/ { print
            substr($0, 3) }' ; } 2>&1 | tee -a "${_full}"

set -x; { echo foo; } 2> file; set +x; echo "file: $(< file)"
```

This section describes how to capture the time used for a command (into a file).

Script `time cmd` prints the execution time of the *cmd* followed to *time*'s *stderr*, not that of *cmd*. We should tell apart the *stdoud* and *stderr* of `time` and `cmd`.

Basically, we group the script with either `{ }` (preferred) or `( )` (sub-shell). Within the group, we capture output (*stdout* and *stderr*) of `cmd` while outside the group, we capture that of `time`.

The second case is `set -x`. Similar to `time`, I want to capture the command executed into file.

11.21.10 tr

Command `tr` translates, squeezes, and/or deletes characters from standard input, writing to standard output. It takes form as:

```
tr [OPTION]... SET1 [SET2]
```

SET1 and/or SET2 define ordered sets of characters like `[:alpha:]` and `c-g` (without brackets). The format of the SET1 and SET2 arguments resembles the format of regular expressions 11.16; however, they are not regular expressions, only lists of characters.

The `-c -C --complement` option replaces SET1 with its complement (all of the characters that are not in SET1). Currently `tr` fully supports only single-byte characters. So characters like Unicode Chinese are not supported.

translate means to substitute each character of its input that is in SET1 to the *corresponding* character in SET2 (required). Therefore, we usually want the lengths of both sets are equal.

To replace comma with newline `tr ',' '\n' < file`. It is quite handy though we can do that with `sed` or `awk`.

A common use of `tr` is to convert lowercase characters to uppercase.

```
tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ

tr a-z A-Z # discouraged method

tr '[:lower:]' '[:upper:]'
```

The bare range `a-z` and `A-Z` are discouraged as it is not portable and depends on implementation.

Option `-d` deletes characters in SET1 and do **not** *translate*. For example `tr -d '\0'` removes all zero bytes.

`-s` can be treated as a special case of `-d`, which *squeeze*s repeated occurence. For example, `tr -s '\n'` merge multiple blank lines to a single one.

Let's have a look at:

```
tr -d -axM # fail

tr -d -- -axM
tr -d axM-
tr -d '[=-=]axM'
```

The code want to delete four characters in which hyphen is included. In the first case, `-a` will be treated as command-line option. The last method uses *equivalence class*. For details, check the `info tr` page.

11.22 Math

Arithmetic in Bash is *integer* math only. To do floating point math, resort to *bc* command.

We call a *math form* as *math context*. The basic math form is `$(())` with the complete syntax at arith expr, which is called Arithmetic Expansion. Basically, within a math context, we use the same syntax as C language like.

```
# POSIX sh
i=$((j + 3))
lvcreate -L "$((24 * 1024))" -n lv99 vg99
q=$((29 / 6)) r=$((29 % 6))
if test "$((a%4))" = 0; then ...
echo "$((2**3))"
```

Within a math context, pay attention to the concept of the *evaluation value* of each *arithmetic expression* and the value returned to the Bash environment by the context itself (name it context value?).

- All C arithmetic operators are supported, including `?:`. Bash brings in *exponentiation* operator `**`.
- Within a single math context, multiple expressions can be separated by *comma*. Value of the last expression becomes that of the math context.

- Variable *name* in a math context are substituted with their values (unset or empty variables are evaluated as 0). There is no need to use parameter expansion (i.e. `${i}`) within math context.

- Numbers without leading 0 are treated as base 10. Numbers with a leading 0x are treated as base 16. Numbers with a leading 0 (not followed by x) are treated as base 8.

11.22.1 Arithmetic Command

Besides, Bash offers two extra forms of math context by *commands* for which, the math context *also* gets an exit status, and side effects. Similar to arithmetic expression `$(())`, the context value of an arithmetic command is that of the command exit code.

Exit code of arithmetic command depends on but not equal to evaluation value of the last expression. If the expression evaluates to 0 and the command is considered a falure. To be logically consistent with Bash, the command returns 1 . Otherwise, the command is regarded successful and return 0.

Consequently,

- Evaluation and boolean logic (0 false, non-zero true) of math expressions are in accord with C syntax.

- Exit code and boolean logic (0 true, non-zero false) of math commands are in accord with Bash.

- Return eithr 0 or 1. Nothing else. This is different to the Arithmetic Expansion above. So we can the Arithmetic Command as a `test` with `for` or `while`.

- Please go back and read Exit Codes and Boolean 11.7.

- Whatever numeric values are involved, boolean logic must be guaranteed. This is the rule that we follow when scripting. Forget about the details then. Check Arithmetic expressions and return codes.

The first arithmetic command is `let`:

```
let a=17+23
echo "a = $a"         # Prints a = 40
#
let a=17 a+=23 a=0    # the last expression evaluates to 0
echo $?               # 1 (false)
#
let a[1]=1+1          # Wrong (if a1=1+1 exists or shopt -s failglob)
( shopt -s failglob; let a[1]=1+1 )
touch a1=1+1; let a[1]=1+1; declare -p a
let 'a[1]=1+1'        # right
```

Note that each arithmetic expression has to be passed as a single argument to the `let` command, so you need quotes if there are spaces or globbing characters.

`let a[1]=1+1` is not right as `[ ]` are glob characters, which matches one and only one of the enclosed characters like regular expression.

> Bash expands globs which appear *unquoted* in commands, by matching *filenames* relative to the current directory. The expansion of the glob results in 1 or more words (0 or more, if certain options are set), and those words (filenames) are used in the command.

So Bash tries to expand `a[1]=1+1` as filename *a1=1+1* before `let` is executed. If that file exists or *failglob* is turned on, Bash reports:

> bash: no match: a[1]=1+1

Now let's move on to the next arithmetic command `(( ))`. It resembles Arithmetic Expansion but removes the leading dollar `$` sign. It is identical to `let` but does not require quotes since expressions inside are delimited by Bash metacharacters `(` and `)`.

```
((a=$a+7))          # Add 7 to a
((a = a + 7))       # Add 7 to a.  Identical to the previous command.
((a += 7))          # Add 7 to a.  Identical to the previous command.

((a = RANDOM % 10 + 1))     # Choose a random number from 1 to 10.
echo $?                     # % is modulus, as in C.

echo "$(( a = RANDOM % 10 + 1 ))" # becomes Arithmetic Expansion
```

Specially, we can compare integers with `(())`. `>` or `<` inside `(( ))` means greater/less than, not output/input redirection involved. Recall that I have talked about expression evaluation and context exit code above. Integer comparision expression also follows the same rules.

The only difference is that arithmetic expression within the command is logic operation: only evaluates to 1 or 0. If the comparison is true (command executes

successfully), the expression evalutes to 1 and returns 0. If it is false, the expression evaluates to 0 and returns 1.

`(( ))` is used more widely than `let`, because it fits so well into an `if` or `while` command like

```
if (( $# > 2 )) ; then printf 'there are more than 2 arguments\n'; fi
```

11.22.2 bc calculator

To do floating point calculation, we use `bc`. However, by default, it truncates according to the *scale* argument instead of rounding.

We increase `scale` and pass the result to `xargs printf` like:

```
bc <<< 'scale=3; 7/242.906' # 0.028
bc <<< 'scale=2; 7/242.906' # 0.02

bc <<< 'scale=3; 7/242.906' | xargs printf "%.2f\n" # 0.03
```

Another possible workaround is to use ± 0.5 trick with the final result like:

```
rounding()
{
    if (( $(bc <<< "$1 < 0") )) ; then offset=-0.5 ; else offset=0.5;
     fi
    printf "%.$2f\n" "$( bc -l <<< 'scale=$2; (((10^$2)*$1)+$offset)
     /(10^$2)' )"
    printf '%.*f\n' "$2" "$( bc -l <<< 'scale=$2; (((10^$2)*$1)+$offset
     )/(10^$2)' )"
}
```

Within the rounding function, we should set the offset according to the sign of the number. If the number is negative, we choose to round it toward the left side of number line. Or we can say to round toward its absolute value. Of course, if we want to always round toward the right side, then just use 0.5.

About `printf '%.*f\n' "$2"`, the asterisk means the width is given as argument before the string or number is printed.

11.23 Redirection

Redirection takes the form as `lhs op rhs`, which can *open*, *duplicate*, *move* or we want to *close* file descriptors.

- *lhs* is always a file descriptor, namely an integer like `0`, `1`, `2`, or `3`. If the *op* is < then there is an *implicit* `0` as `0<`. If it's > or >>, there is an *implicit* `1` as `1>` or `1>>`.
- *op* is <, >, >>, >|, or <>. Two special redirections << (here document) and <<< (here string) usually require string(s) for *rhs*. Details, read the official manual.
- *rhs* is the thing that the file descriptor will describe. It can be a filename, or the place where another descriptor goes (prefixed with `&` like `&1`), or `&-` that will close the *lhs* file descriptor.

When redirection is used, the *lhs* is pointed to what *rhs* is **currently** pointed to. If, later on, *rhs* is pointed to another place, *lhs* remains and won't follow *rhs*'s update. We can think of a file descriptor as C language pointer.

To check which place file descriptors are currently pointed, we can:

```
ll /proc/$$/fd/

# total 0
# dr-x------ 2 outsinre outsinre  0 Feb 14 15:49 .
# dr-xr-xr-x 9 outsinre outsinre  0 Feb 14 15:49 ..
# lrwx------ 1 outsinre outsinre 64 Feb 14 15:49 0 -> /dev/pts/2
# lrwx------ 1 outsinre outsinre 64 Feb 14 15:49 1 -> /dev/pts/2
# lrwx------ 1 outsinre outsinre 64 Feb 14 15:49 2 -> /dev/pts/2
# lrwx------ 1 outsinre outsinre 64 Feb 14 21:10 255 -> /dev/pts/2

for fd in 0 1 2 255; do cat /proc/$$/fdinfo/$fd; echo; done
```

In the following code, *cmd* reads input from filename *myFile*. File descripted `3` is associated with `1` (standard output, */dev/pts/2*). But the script does not make use of the new descriptor. `2` (standard error, */dev/pts/2*) is redirected to `/dev/null`. Standard output (implicit `1`, */dev/pts/2*) is redirected to `2`, which is now pointed to */dev/null*.

```
# Good! This is clearly a simple commmand with two arguments and 4
    redirections
cmd arg1 arg2 <myFile 3<&1 2>/dev/null >&2
```

There are two special redirection forms `fd1>&fd2` and `fd1<&fd2`. As mentioned earlier, if *fd1* is omitted, the default value are `1` and `0` respectivelly.

They are called *descriptor copy* or *descriptor duplication*. Technically speaking, the two forms **make no difference**. Yeah, they are equal in Bash grammar except the different implicity *lhs*.

fd1 is opened (created) if it does not exist, and pointed to where *fd2* is currently pointed. Whether *fd1* is opened for reading or writing, depends on *fd2*. If the file *fd2* linked with is on read mode, then we can use *fd1* for reading. Similarly, we can write to *fd1* when *fd2* points to a file on write mode. Obviously, if the file is on read/write mode, *fd1* can be used to read from and write to that file.

In a script, we can do like this:

```
exec m>&n
exec m<&n
# -or-
cmd arg1 arg2 m>&n
cmd arg1 arg2 m<&n
```

It is necessary to tell apart `>,` `<` and `>&,` `<&` as the `&` requires an integer descriptor followed while the former needs a filename. More importantly, `>,` `<` cares about the read and write mode. `>` opens a file descriptor for writing (redirecting output). The other one for reading (redirecting input). Apparently, `>>` is for appending redirected output.

To open a file descriptor for both reading and writing, we can:

```
[n]<>word

exec 3<>/path/to/filename
```

If *n* is omitted, it defaults to 0. If the filename of *word* expansion does not exist, it is created.

Sometimes, we need to store the integer value in a variable. Then enclose it with braces:

> Each redirection that may be preceded by a file descriptor number may instead be preceded by a word of the form `{varname}`. In this case, for each redirection operator except `>&-` and `<&-`, the shell will allocate a file descriptor greater than or equal to 10 and assign it to varname. If `>&-` or `<&-` is preceded by `{varname}`, the value of varname defines the file descriptor to close.

```
fd=0; echo "hello, world" >> /tmp/foo; exec {fd}</tmp/foo;
printf '%d\n' $fd
read -r -u "$fd" line; printf '%s\n' "$line"
```

We can also *move* a file descriptor, which first duplicate the *rhs* and then close it.

```
[n]<&digit-
[n]<&digit-

m<&n-; m>&n-
<&4-; 0<&4-
>&4-; 1>&4-
```

Read more at Switch stdout and stderr, What does `3>&1 1>&2 2>&3` do in a script and Closing a file descriptor, `>&` vs. `<&-`.

11.24 Set or Not

It happens when we want to test whether a *name* is set or not. Just use:

```
[[ "${array[key]+abc}" ]] && echo "exists"
```

This code applies to both indexed and associative array. `"${array[key]+abc}"` is actually a special form of parameter expansion with the colon omitted. The original form is `${parameter:+word}`. From the manual, we find:

> if the colon is omitted, the operator tests only for existence [of parameter]

- if `array[key]` is set, return *abc*.
- if `array[key]` is not set, return nothing

However, we still can add the colon but the substitution procedure is carried out, though that does not affect the logic.

```
# degrade performance

[[ "${array[key]:+abc}" ]] && echo "exists"
```

Another method is to use built-in *test* option `-v VAR`:

-v VAR True if the shell variable VAR is set.

When to check for existence in this method, do **not** use parameter expansion. The name itself is enough.

```
[[ -v ar["hello"] ]] && echo "exists"
declare a=1; [[ -v a ]] && echo "exists" || echo "no"

declare -A ar=( [0]=1 ['b c']=2 ); [[ -v ar ]] && echo "yes"
declare -A ar=( [a]=1 ['b c']=2 ); [[ -v ar[@] ]] && echo "yes"
```

The first two lines test whether a name set.

The third tests the key of `0`. In the 4th case, the bare `a[@]` (without `$` prefixed) is **not** a parameter expansion and hence it does **not** expand to postional index. This code tests for any array element. If there exists at least one element set, then exits true.

A different version is:

```
# test var
_regex="^declare -[aA] ${var}[=|$]"
[[ "$(declare -p $ar)" =~ "${_regex}" ]] && echo "yes"
```

Here is a function to test whether a specific value is stored in array. It is mainly for index arrays. If it is an associative array, just use the corresponding key to test.

```
#!inarray
# Usage: inarray "$value" "${array[@]}"
inarray() { local n=$1 h; shift; for h; do [[ $n = "$h" ]] && return;
    done; return 1; }
```

11.25 String as Delimiter

To split string with just a single delimter, we just need `read` command and `IFS` variable.

In the section 11.20.3, `mapfile`, `awk`, and *process substitution* are used to split a string with another string. I will introduce Split strings with group delimiters in this section.

```
_headers_sep=')@|#('
_headers_regex=$'\^\~\$headers=\'(.*)\)@\|#\(\'' #'

tmp_headers="${BASH_REMATCH[1]}${_headers_sep}" _headers_array=()
while [[ -n $tmp_headers ]]
do
    _headers_array+=( "${tmp_headers%%${_headers_sep}*}" )
    tmp_headers="${tmp_headers#*${_headers_sep}}"
done; unset tmp_headers
```

We define the string delimiter as `_headers_sep`. When doing the parameter expansion, there is no need to escape those characters as `_headers_regex` does.

Part VI

Networking

Chapter 12

Data Bills

12.1 95th Percentile

95th percentile is a commonly used commercial scheme to calculate and evaluate the regular and sustained used of a network connection. It is a kind of Burstable Billing that measures bandwidth based on *peak use*.

Generally, it allows the usage to exceed a specified threshold for a short period of time without the financial penalty of purchasing a higher Committed Information Rate (CIR is another billing method) from an ISP.

There are two critical factors involved in this scheme, namely the *percentile* and the *sampling interval*. Usually, they are 95 and 5 respectively and therefore it is also called *95/5 percentile*.

Given a monthly billing cycle, 95/5 percentile allows a customer to have a short (i.e. threshold is $30 \times 24 \times 5\% = 36$ hours) burst in in traffic without overage charges. That is to say, 95% of the time, the usage is below this ammount. Conversely, 5% of the samplings may be bursting above this rate.

How is *this ammount* or *this rate* is calculated? Usually a time-bandwidth chart is drawn to visualize bandwidth usage through a bill cyle like figure 12.1 whose *integral area* reflects the total data transmitted.

Bandwidth is sampled at an interval of 5 minutes from the switch or router. During an interval, the average bandwidth is calculated as the number of bits transferred throughout the interval divided by the duration (5m = 300s).

At the end of the month, all the bandwidth samples ($30 \times 24 \times 60 \div 5 = 8640$) are sorted from highest to lowest. The top 5% ($8640 \times 5\% = 432$) samples is thrown away and the next highest sample 433 becomes the billable use for the entire month.

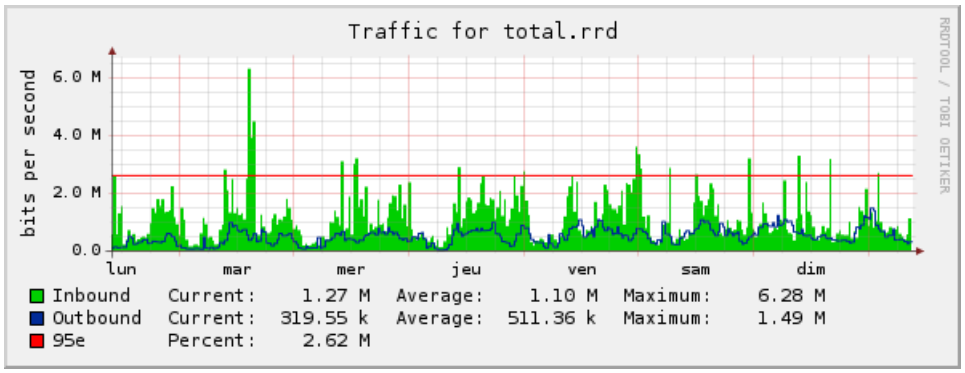


Figure 12.1: 95/5 Percentile

Obviously, $432 \times 5 \div 60 = 36$. More specifically, bandwidth could be used at a higher rate for up to $24 \times 60 \times 5\% = 72$ minutes a day with no financial penalty.

In the figure 12.2, the billable value falls around 6Mbps, or 60% of the highest burst. From this figure, we find the sorted slope is quite steep. The sharp burst contributes a lot to the final bill though not much bandwidth is used.

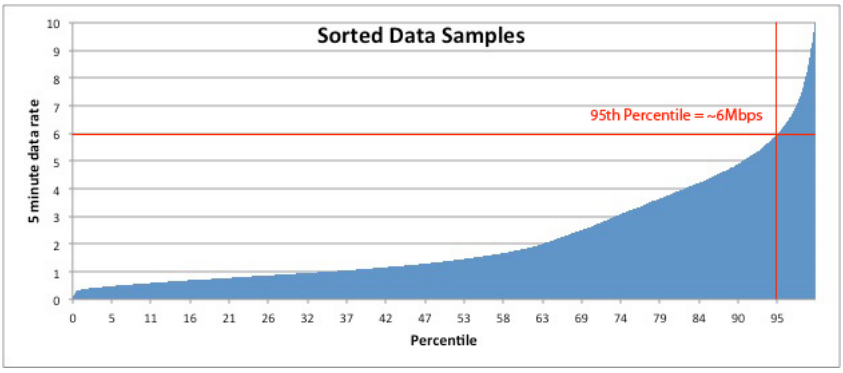


Figure 12.2: Sorted Percentile

Conversely, at any moment during the billing cycle, even if peak traffic only appears for a short *instant* and no additional traffic is generated, the billing ammount can be sbustantially higher than average usage billing.

This is not the end of the story. Although we get the billable value, it may not take affect when paying bills. For example, what if all customers' 95th percentile valie are pretty small? In that case, the ISP's infrastructure would be wasted and lose money.

95th percentile also assumes a *base commit rate* (and fees associated) that must be guranteed by customers. If the 95th percentile value only take effect when it is

higher than the base commit rate, otherwise the later value is used irrespective of the real 95th percentile value. The higher the base commit rate is, the cheaper the bandwidth fee becomes.

Base commit rate is *not* CIR where bandwidth must be reserved for a customer but just a billing gurantee even the real bandwidth usage is 0.

Base commit rate is fairly to both the carrier and the customer in terms of the service delivered to the customer and the ability of the carrier to scale its infrastructure to meed different customers' needs.

12.2 Other Billing Methods

The 95th percentile is utilized mostly between ISP and business customers in a datacenter while CIR is adopted between ISP and individuals for home cable, fiber, DSL etc. Hard bandwidth is guranteed and no burst is allowed. Customers pay for the fixed bandwidth value.

Apart from 95th percentile and CIR, *actual throughput billing* is used in limited and shared bandwidth networks like mobile data networks (i.e. 4G) where where resources overprovisioning is not possible due to limited spectrum availability. Customers just pay for what they get on demand. ISP will simply record how much data you moved over the circuit for that interval. It is also used by some web-based VPS platforms to bill customers.

You may think of *average rage bill* where customers pay for their average committed bandwidth. However, this scheme requires infrastructure overprovisioning on the part of ISP to meet customers' peak bandwidth usage. It is not used in real world!

You can read more about billing methods on billing methods explained and analyzed.

Part VII

Brainteaser

Chapter 13

Brain Teasers

补全数列空缺

例 3 5 10 25 75 () 875. 观察此数列发现，相邻两项的数间没有什么特别关系，相乘相加都不是。这时应考虑相邻三、四项项的关系，会发现 $3 \times 5 + 10 = 25$, 类似地有 $5 \times 10 + 25 = 75$. 因此空处的值应是 $10 \times 25 + 75 = 325$. 再来一个例子，5 6 16 28 60 (). 分析得出 $5\ times 2 + 6 = 16$, 还有 $6 \times 2 + 16 = 28$. 所以空处的值应是 $28 \times 2 + 60 = 116$. 此两例考虑连续多个数之间的递推关系，类似于 Fibonacci Sequence.

牛吃草

一块匀均生长的草地，可供 27 头牛吃 6 周，或 23 牛吃 9 周，问多少牛可吃 18 周？

我们先设一头牛吃草的速度是 a, 单位可以是 kg/w, 则 6 周吃 $27 \times 6 \cdot a$ kg 的草。设草的生长速度是 b, 单位可以是 kg/w, 那么 6 周长了 $6 \cdot b$ kg 的草。那么是不是 $27 \times 6 \cdot a = 6b$ 呢？

不是！因为牛吃草前，草地上本来就有部分草，设为 c kg, 则 $27 \times 6 \cdot a = 6b + c$, 同理我们得出：

$$
\begin{aligned}
27 \times 6 \cdot a &= 6 \cdot b + c \\
23 \times 9 \cdot a &= 9 \cdot b + c \\
x \times 18 \cdot a &= 18 \cdot b + c
\end{aligned}
$$

通过前两个方程，可得出 $b = 15 \cdot a$ 和 $c = 72 \cdot a$, 进而求出 $x = 19$.

相向而行

甲乙两地相距 90 米，A, B 二人分别从甲乙同时出发，在两地来回跑动。甲的速度是 3 m/s, 乙的是 2 m/s. 问 10 分钟内甲乙相遇几次？

第一次相遇时二人行使 90 米，时间是 $90 \div (3+2) = 18$s. 相遇后二人背向而行，直至达到对端。此时二人又行使了 90 米，用时 18s.

二人再次相向而行, 准备下一次相遇。此后一直重复此过程。可以发现相遇一次二人行使了 180 米，花时 36s, 所以 10 分钟相遇次数是 $10 \times 60 \div 36 = 16\frac{2}{3}$, 结果是分数，那么是 16 次还是 17 次呢？

应是 17 次，因为第次相遇时，在 36s 的前半程，所以分数部分应四舍五入!

上面这种方法不够明了。第一次相遇需 18s, 下一次相遇需 36s, 再下一次相遇需 36s, 此后每一次相遇都需 36s. 我们找相遇的时间点更合理，得到一个等差数列:

$$
\begin{aligned}
t_1 &= 18 \\
t_2 &= 18 + 36 \\
&... \\
t_n &= 18 + (n-1) \cdot 36
\end{aligned}
$$

根据 $t_n = 10 \times 60$ 可以算出 $n = 17\frac{1}{6}$. 由此可知，相遇 17 次，因为剩下的 1/6 还没到第 17 次相遇时间点。

背向而行

有一 300 米圆跑道，甲乙在同一地点背向而行，速度分别是 3.5m/s 和 4m/s. 问第 10 次相遇时，甲离出发地还有多远？

每相遇一次，两人共行走了 300 米，其中甲行走了 $\frac{3.5}{(3.5+4)} \times 300 = 140$ 米，而乙行走了 160 米。

那么相遇 10 次时, 甲累计走了 10×1401400 米, 而 $1400 \div 300 = 4 \cdots\cdots 200$, 也就是说走了 4 圈后，第 5 圈走了 200 米，所以离出发点还剩 100 米。

交换变量

有两个整数 a = 5, b = 7, 不通过临时变量，如何交换二者的值？

方法一：异或 XOR. 异或的意思是，让两个数的二进制位互相比对，同为 0 或 1 时结果为 0, 有一个 1 一个 0 （相异）时结果为 1. 所以任何数异或自己的结果是 0. 对应的 C 语言结果是:

```
a = a^b;
b = a^b;
a = a^b;
```

Listing 13.1: Swap by XOR

方法二：加减法。

```
a = a + b;
b = a - b;
a = a - b;
// Or
a = (a + b) - (b = a)
```

Listing 13.2: Swap by Arithmetic Operations

Chapter 14

行测图行推理

14.1 规律总结

首先分析图形外观，考察图形画法，得出图形里的分部要素，也可称为元素。

1. 边。三角形、四边形、五边形……
2. 曲线。
3. 圆。
4. 区。图形被分割成不同的封闭区域。
5. 交点。边上的交点。

找出各图形元素后，再分析其规律。

1. 行列。从行和列两个维度来分析规律。
2. 数量。每种元素的数量可能呈某种关系，如相等，差数列。还可以是行、列上数量关系。如边数，交点数，区域数等。
3. 笔画。是否是一笔画完。
4. 对称。如轴对称（竖轴、横轴），旋转对称（中心对称，180 度），不对称。如英文字母的对称性。还可能数对称轴（相同的、不同的）的数量。
5. 旋转。图形整体依次顺时针、逆时针旋转一个角度。
6. 翻转。图形整体以某轴翻转 180 度。如水平翻转。
7. 平移。某元素在图内平移一定距离。如小方格每次移顺时针移 3 个位置。

8. 相对。不同元素在图内相对位置发生变化。相对位置可以是内部、外部，相交、相接、分离，也可以是直角处、圆弧处，还可以是在最长边、最短边处。
9. 加减。相邻图形或元素之间加减组合出新图。如第 1 个图形和第 2 个图形组合起来是第 3 个图形。常见的是九宫格里，行、列间图形是加减关系。有时甚至是先旋转再加减。
10. 类别。图形可以按其属性分类。如同为生活类用品，同属用腿、用手、手脚并用的运动项目。

Part VIII

Chemistry

Chapter 15

走进化学世界

化学就是研究物质及其变化，它不仅研究已经存在的物质，还要研究和创造自然界原本不存在的新物质。例如，半导体材料，电阻几乎为零的超导体，有记忆能力的新材料，等等。

化学在保证人类生存和提高生活质量上有很大帮助。例如：

- 化肥和农药，增加粮食产量。
- 化学合成药物，指抑制细菌和病毒。
- 化学新能源、新材料。

近代化学突破源于两点：

1. 物质是由分子原子构成的，分子中原子的重新组合是化学变化的基础。
2. 元素周期表。

这两个突破使得化学研究有迹可寻。化学的科学定义：**化学是在分子、原子层上研究物质性质、组成、结构与变化规律的科学**。研究物质是指研究其性质、组成和结构等静态特征，而变化是化学反应，也即原子的重新组合，这是一个动态过程。

15.1 性质及变化

物质变化分两两种：

- 物理变化：没有生成新物质的变化，如物质形态发生变化。气态、液态、固态三态之间的相互转换的过程就是物理变化。
- 化学变化或叫化学反应：生成新物质的变化（原子重新组合），常表现为颜色变化，放出气体，生成沉淀等。化学变化还伴随物质能量变化，

如吸热、放热、发光等。

虽然物理变化也通常伴随能量转换，但主要借用外部能量。而化学变化和能量可以只来自物质本身，不需要外部能量辅助。

依据物质变化过程中表现出的性质，可有：

- 化学性质：物质在化学变化中表现出的性质。例如，铜在潮湿的空气中生成铜绿。
- 物理性质：物质不需要化学反应就表现出来的性质。物质颜色、状态、气味、硬度、熔点、沸点、密度等都是物理性质。例如，常态下，氧气是种无色、无味的气体。注意，物理性质可能是物理变化过程表现出的性质，如物质的三态。

外界条件改变时，物质性质也会随着变化，因此，描述物质性质时往往要注明条件。如，当温度升高时，固态冰变成液态水，再加温，水会沸腾。液体的沸点是物理性质，但受大气压强的影响。大气稀薄的地方，大气压强变小，这时水的沸点会降低，容易烧开。

15.1.1 大气压

由于大气压强是变化的，人们把 $1atm = 101kPa = 76cm$水银柱重量 当作标准大气压强。大气压强是指大气对浸在它里面的物体产生的压强，也叫大气压或气压，可以用空气的重力或分子热运动来解释其产生机理，而且这两种解释是等价的。

那么如何用重力和分子热运动解释呢？在密封空间内，气压的产生应从微观上来解释。气体分子热运动时，撞击空间内壁，产生作用力（内壁也同时产生反作用力），这个作用力就是气体压力。没有内壁就不会产生气体压力！压强是单位面积上的压力，表示强度，不用考虑内壁的存在。

由于分子作用力各向同性，所以任一点的压强在不同方向上相同。

对于空气呢？空气分子同样作热运动，但是没有内壁，怎么产生压力呢？有地球引力！地球引力相当于上述密闭空间内壁的反作用力。所以空气中任意一点也有空气压力，进而也有压强。同理，空气压力也是各向同性的。

至于压强的数值计算，两种情况有所不同，关键是算出分子热运动的作用力。对于密闭空间，它是 $P = nTR/volume$. 对于空气，因为重力和空气分子热运动产生的撞击力平衡，大气压强是大气施加于单位面积上的重力，即该地单位面积垂直向上延伸到大气层顶的空气柱的总重力。简单的数学公式是 $P = m \cdot g/area$. 实际中要考虑不同高度处重力加速度 g 的不同。

15.2 化学实验

15.2.1 注意事项

- 手不接触药品，不尝药品，鼻孔不可太靠近瓶口（特别是气体）。
- 实验剩余药品不能放回原瓶，不能随意丢弃，不能带出实验室。要放入指定容器。
- 节约药品。未说明剂量时，液体一般取 $1-2\,mL$, 固体只需盖满试管底部即可。
- 保护眼睛，如果进了药液，应立即用清水清洗，要眨眼睛。

15.2.2 药品取用

- 固体：广口瓶。药匙（粉状、颗粒）、摄子（块状），取完应擦净。玻璃容器横放，块状放容器口，缓缓竖立，滑入底。药匙送粉状入试管底，再直立。
- 液体：细口瓶。倾倒法。瓶塞倒立桌面，瓶口紧挨试管口，瓶身标签面朝手心。定量取液，用量筒。量液时，视线与凹液面最低处保持水平。仰视偏多，俯视偏少。取少量液体用滴管，应保持橡胶帽朝上，不可平放、倒放，否则腐蚀橡胶帽或污染试剂。滴管应悬空滴液，不可接触容器口，否则因为试剂太少，沾在内壁。用完即洗。

15.2.3 物质加热

一般用酒精灯。不可以向燃着的酒精灯添加酒精，也不可以用一个点燃另一个。用灯帽熄灭，不可用嘴吹。熄灭后，取下灯帽再盖上。

加热试管液体：

1. 试管外壁应干燥，液体不超过容积的 1/3.
2. 试管夹由试管底部套上、取下。
3. 先使试管底部均匀受热，再用外焰固定加热。
4. 试管口不要对着自己或他人。
5. 加热后的试管不能立即接触冷水或用冷水冲洗。

三层火焰

本节顺便说下蜡烛然烧问题。蜡烛然烧时，蜡固体先熔化、气化，再与空气中氧气发生化学反应。

外焰是红白色，温度高；而内焰为红色且边缘是蓝色，温度低。焰心没有发生燃烧，所以不烧手。温度的高低主要由与氧气接触面积决定。外焰处氧气最多，燃烧最充分，温度最高。焰心处氧气已消耗殆尽，所以没有燃烧。

色温和温度是两个不同的概念。色温反应的是单个分子的能量，能量越高，颜色越偏蓝。温度是分子热运动的宏观测量，不仅要考虑单个分子能量，还应考虑分子总个数。在相同分子个数情况下，蓝焰温度肯定比黄色或红色高。

但在蜡烛然烧时，外焰温高除了氧气多外，和热气体上升也有关系。另外，既然外焰燃烧更充分，为何不见蓝色。实际是有蓝色的，只是被遮住了。外焰处蜡蒸气非常多，被外焰加热后，变成白炽色，把燃烧的蓝色吸收了。所以外焰的高温不是因为黄色，而是蓝色（充分燃烧）和上升热气。

15.2.4 仪器连接

玻璃管，胶皮管，橡胶塞：

1. 玻璃管和橡胶塞：玻璃管口用水湿润，对准橡胶塞上的孔稍用力转动、插入。
2. 玻璃管和胶皮管：玻璃管口用水湿润，稍用力即可插入。
3. 容器口和橡胶塞：慢慢转动橡胶塞，塞入容器口（如试管）。
4. 气密性检查：用手握紧试管，观察水中导管口是否有汽泡冒出，有则好。

15.2.5 洗涤玻璃容器

必需洗涤玻璃容器，否则影响实验效果。以洗试管为例：

1. 倒掉废液。
2. 注入半试管水，振荡后再倒掉。
3. 重复上一步聚。
4. 如试管内壁还有不易洗掉残留物质，用试管刷刷洗。洗刷时，须转动或上下移动试管刷。

洗过的玻璃容器内壁附着的水既不聚成水滴，也不股下流，表明仪器已洗干净。

Chapter 16

空气

16.1 简介

空气的主要成分是氮 78%, 氧 21%, 稀有气体 0.94%, 二氧化碳 0.03%, 其它气体和杂质占 0.03%. 其中氮和氧几乎各占 1/5 和 4/5. 稀有气体主要是氦 hài, 氖、氩、氪、氙 xiān, 氡等。

- 混合物：由两种或两种以上的物质混合而成的物质。组成混合物的各种成分保持着它们各自的性质。
- 纯净物：只有一种物质组成。纯净物可以用化学式表示，如氮气是 N_2, 磷是 P, 五氧化二氮是 P_2O_5 等。

16.2 成分

各主要成分。

Chapter 17

物质列表

叁数 / 化学式	名称、别名	颜色和状态
$CuSO_4$	硫酸铜、胆矾、蓝矾	（无水）灰白色粉末、（有水）蓝色结晶固体
H_2O	水	无色液体
$KAl(SO_4)\cdot 2\,H_2O$	十二水合硫酸铝钾、明矾	无色或白色的八面体晶体
Fe	铁	银白色固体
Al	铝	银白色固体
O_2	氧气	无色无味气体

Table 17.1: 常见物质列表

Periodic Table of the Elements

1 2.20 1s																	2 1s
H Hydrogen 1.00784–1.00811																	**He** Helium 4.002602(2)
3 0.98 2s	4 1.57 2s											5 2.04 2p	6 2.55 2p	7 3.04 2p	8 3.44 2p	9 3.98 2p	10 2p
Li Lithium 6.938–6.997	**Be** Beryllium 9.0121831(5)											**B** Boron 10.806–10.821	**C** Carbon 12.0096–12.0116	**N** Nitrogen 14.00643–14.00728	**O** Oxygen 15.99903–15.99977	**F** Fluorine 18.998403163(6)	**Ne** Neon 20.1797(6)
11 0.93 3s	12 1.31 3s											13 1.61 3p	14 1.90 3p	15 2.19 3p	16 2.58 3p	17 3.16 3p	18 3p
Na Sodium 22.98976928(2)	**Mg** Magnesium 24.304–24.307											**Al** Aluminium 26.9815385(7)	**Si** Silicon 28.084–28.086	**P** Phosphorus 30.973761998(5)	**S** Sulphur 32.059–32.076	**Cl** Chlorine 35.446–35.457	**Ar** Argon 39.948(1)
19 0.82 4s	20 1.00 4s	21 1.36 3d	22 1.54 3d	23 1.63 3d	24 1.66 3d*	25 1.55 3d	26 1.83 3d	27 1.88 3d	28 1.91 3d	29 1.90 3d*	30 1.65 3d	31 1.81 4p	32 2.01 4p	33 2.18 4p	34 2.55 4p	35 2.96 4p	36 3.00 4p
K Potassium 39.0983(1)	**Ca** Calcium 40.078(4)	**Sc** Scandium 44.955908(5)	**Ti** Titanium 47.867(1)	**V** Vanadium 50.9415(1)	**Cr** Chromium 51.9961(6)	**Mn** Manganese 54.938044(3)	**Fe** Iron 55.845(2)	**Co** Cobalt 58.933194(4)	**Ni** Nickel 58.6934(4)	**Cu** Copper 63.546(3)	**Zn** Zinc 65.38(2)	**Ga** Gallium 69.723(1)	**Ge** Germanium 72.630(8)	**As** Arsenic 74.921595(6)	**Se** Selenium 78.971(8)	**Br** Bromine 79.901–79.907	**Kr** Krypton 83.798(2)
37 0.82 5s	38 0.95 5s	39 1.22 4d	40 1.33 4d	41 1.6 4d*	42 2.16 4d*	43 1.9 4d	44 2.2 4d*	45 2.28 4d*	46 2.20 4d*	47 1.93 4d*	48 1.69 4d	49 1.78 5p	50 1.96 5p	51 2.05 5p	52 2.1 5p	53 2.66 5p	54 2.60 5p
Rb Rubidium 85.4678(3)	**Sr** Strontium 87.62(1)	**Y** Yttrium 88.90584(2)	**Zr** Zirconium 91.224(2)	**Nb** Niobium 92.90637(2)	**Mo** Molybdenum 95.95(1)	**Tc** Technetium (98)	**Ru** Ruthenium 101.07(2)	**Rh** Rhodium 102.90550(2)	**Pd** Palladium 106.42(1)	**Ag** Silver 107.8682(2)	**Cd** Cadmium 112.414(4)	**In** Indium 114.818(1)	**Sn** Tin 118.710(7)	**Sb** Antimony 121.760(1)	**Te** Tellurium 127.60(3)	**I** Iodine 126.90447(3)	**Xe** Xenon 131.293(6)
55 0.79 6s	56 0.89 6s	57-71 *	72 1.3 5d	73 1.5 5d	74 2.36 5d	75 1.9 5d	76 2.2 5d	77 2.20 5d	78 2.28 5d*	79 2.54 5d*	80 2.00 5d	81 1.62 6p	82 1.87 6p	83 2.02 6p	84 2.0 6p	85 2.2 6p	86 2.2 6p
Cs Cesium 132.90545196(6)	**Ba** Barium 137.327(7)	Lanthanides	**Hf** Halfnium 178.49(2)	**Ta** Tantalum 180.94788(2)	**W** Tungsten 183.84(1)	**Re** Rhenium 186.207(1)	**Os** Osmium 190.23(3)	**Ir** Iridium 192.217(3)	**Pt** Platinum 195.084(9)	**Au** Gold 196.966569(5)	**Hg** Mercury 200.592(3)	**Tl** Thallium 204.382–204.385	**Pb** Lead 207.2(1)	**Bi** Bismuth 208.98040(1)	**Po** Polonium (209)	**At** Astatine (210)	**Rn** Radon (222)
87 0.7 7s	88 0.9 7s	89-103 **	104 6d	105 6d	106 6d	107 6d	108 6d	109 6d	110 6d	111 6d	112 6d	113 7p	114 7p	115 7p	116 7p	117 7p	118 7p
Fr Francium (223)	**Ra** Radium (226)	Actinides	**Rf** Rutherfordium (261)	**Db** Dubnium (268)	**Sg** Seaborgium (269)	**Bh** Bohrium (270)	**Hs** Hassium (269)	**Mt** Meitnerium (278)	**Ds** Darmstadtium (281)	**Rg** Roentgenium (282)	**Cn** Copernicium (285)	**Nh** Nihonium (286)	**Fl** Flerovium (289)	**Mc** Moscovium (289)	**Lv** Livermorium (293)	**Ts** Tennessine (294)	**Og** Oganesson (294)

*	57 1.1 5d* **La** Lanthanum 138.90547(7)	58 1.12 4f* **Ce** Cerium 140.116(1)	59 1.13 4f **Pr** Praseodymium 140.90766(2)	60 1.14 4f **Nd** Neodymium 144.242(3)	61 1.13 4f **Pm** Promethium (145)	62 1.17 4f **Sm** Samarium 150.36(2)	63 1.2 4f **Eu** Europium 151.964(1)	64 1.2 4f* **Gd** Gadolinium 157.25(3)	65 1.1 4f **Tb** Terbium 158.92535(2)	66 1.22 4f **Dy** Dysprosium 162.500(1)	67 1.23 4f **Ho** Holmium 164.93033(2)	68 1.24 4f **Er** Erbium 167.259(3)	69 1.25 4f **Tm** Thulium 168.93422(2)	70 1.1 4f **Yb** Ytterbium 173.045(10)	71 1.27 4f **Lu** Lutetium 174.9668(1)
**	89 1.1 6d* **Ac** Actinium (227)	90 1.3 5f* **Th** Thorium 232.0377(4)	91 1.5 5f* **Pa** Protactinium 231.03588(2)	92 1.38 5f* **U** Uranium 238.02891(3)	93 1.36 5f* **Np** Neptunium (237)	94 1.28 5f **Pu** Plutonium (244)	95 1.13 5f **Am** Americium (243)	96 1.28 5f* **Cm** Curium (247)	97 1.3 5f **Bk** Berkelium (247)	98 1.3 5f **Cf** Californium (251)	99 1.3 5f **Es** Einsteinium (252)	100 1.3 5f **Fm** Fermium (257)	101 1.3 5f **Md** Mendelevium (258)	102 1.3 5f **No** Nobelium (259)	103 1.3 5f **Lr** Lawrencium (266)

Z = atomic number; eneg = electronegativity; ss = subshell; Sy = Symbol, Name = element name, saw = standard atomic weight

Standard atomic weights taken from the Commission on Isotopic Abundances and Atomic Weights (ciaaw.org/atomic-weights.htm). Adapted from Ivan Griffin's LaTeX Periodic Table.

An asterisk (*) next to a subshell indicates an anomalous (Aufbau rule-breaking) ground state electron configuration.

English-Chinese Periodic Table of Elements 英漢元素周期表

1 / Ia																	18 / VIIIa
1 氫 **H** 1 hydrogen 1.0079 1s¹	2 / IIa											13 / IIIa	14 / IVa	15 / Va	16 / VIa	17 / VIIa	2 氦 **He** helium 4.0026 1s²
3 鋰 **Li** 1 lithium 6.941 [He]2s¹	4 鈹 **Be** 2 beryllium 9.0122 [He]2s²											5 硼 **B** 3 boron 10.811 [He]2s²2p¹	6 碳 **C** 4,2 carbon 12.011 [He]2s²2p²	7 氮 **N** ±3,5,4,2 nitrogen 14.007 [He]2s²2p³	8 氧 **O** -2 oxygen 15.999 [He]2s²2p⁴	9 氟 **F** -1 fluorine 18.998 [He]2s²2p⁵	10 氖 **Ne** neon 20.180 [He]2s²2p⁶
11 鈉 **Na** 1 sodium 22.990 [Ne]3s¹	12 鎂 **Mg** 2 magnesium 24.305 [Ne]3s²											13 鋁 **Al** 3 aluminum 26.982 [Ne]3s²3p¹	14 硅 **Si** 4 silicon 28.086 [Ne]3s²3p²	15 磷 **P** ±3,5,4 phosphorus 30.974 [Ne]3s²3p³	16 硫 **S** ±2,4,6 sulfur 32.065 [Ne]3s²3p⁴	17 氯 **Cl** ±1,3,5,7 chlorine 35.453 [Ne]3s²3p⁵	18 氬 **Ar** argon 39.948 [Ne]3s²3p⁶
19 鉀 **K** 1 potassium 39.098 [Ar]4s¹	20 鈣 **Ca** 2 calcium 40.078 [Ar]4s²	21 鈧 **Sc** 3 scandium 44.956 [Ar]4s²3d¹	22 鈦 **Ti** 4 titanium 47.867 [Ar]4s²3d²	23 釩 **V** 5,3 vanadium 50.942 [Ar]4s²3d³	24 鉻 **Cr** 6,3,2 chromium 51.996 [Ar]4s¹3d⁵	25 錳 **Mn** 7,6,4,2,3 manganese 54.938 [Ar]4s²3d⁵	26 鐵 **Fe** 2,3 iron 55.845 [Ar]4s²3d⁶	27 鈷 **Co** 2,3 cobalt 58.933 [Ar]4s²3d⁷	28 鎳 **Ni** 2,3 nickel 58.693 [Ar]4s²3d⁸	29 銅 **Cu** 2,1 copper 63.546 [Ar]4s¹3d¹⁰	30 鋅 **Zn** 2 zinc 65.409 [Ar]4s²3d¹⁰	31 鎵 **Ga** 3 gallium 69.723 [Ar]4s²3d¹⁰4p¹	32 鍺 **Ge** 4 germanium 72.64 [Ar]4s²3d¹⁰4p²	33 砷 **As** ±3,5 arsenic 74.922 [Ar]4s²3d¹⁰4p³	34 硒 **Se** ±2,4,6 selenium 78.96 [Ar]4s²3d¹⁰4p⁴	35 溴 **Br** ±1,5 bromine 79.904 [Ar]4s²3d¹⁰4p⁵	36 氪 **Kr** krypton 83.798 [Ar]4s²3d¹⁰4p⁶
37 銣 **Rb** 1 rubidium 85.468 [Kr]5s¹	38 鍶 **Sr** 2 strontium 87.62 [Kr]5s²	39 釔 **Y** 3 yttrium 88.906 [Kr]5s²4d¹	40 鋯 **Zr** 4 zirconium 91.224 [Kr]5s²4d²	41 鈮 **Nb** 5,3 niobium 92.906 [Kr]5s¹4d⁴	42 鉬 **Mo** 6,5,4,3,2 molybdenum 95.94 [Kr]5s¹4d⁵	43 鎝 **Tc** 7 technetium [98] [Kr]5s²4d⁵	44 釕 **Ru** 2,3,4,6,8 ruthenium 101.07 [Kr]5s¹4d⁷	45 銠 **Rh** 2,3,4 rhodium 102.91 [Kr]5s¹4d⁸	46 鈀 **Pd** 2,4 palladium 106.42 [Kr]4d¹⁰	47 銀 **Ag** 1 silver 107.87 [Kr]5s¹4d¹⁰	48 鎘 **Cd** 2 cadmium 112.41 [Kr]5s²4d¹⁰	49 銦 **In** 3 indium 114.82 [Kr]5s²4d¹⁰5p¹	50 錫 **Sn** 4,2 tin 118.71 [Kr]5s²4d¹⁰5p²	51 銻 **Sb** ±3,5 antimony 121.76 [Kr]5s²4d¹⁰5p³	52 碲 **Te** ±2,4,6 tellurium 127.60 [Kr]5s²4d¹⁰5p⁴	53 碘 **I** ±1,5,7 iodine 126.90 [Kr]5s²4d¹⁰5p⁵	54 氙 **Xe** xenon 131.29 [Kr]5s²4d¹⁰5p⁶
55 銫 **Cs** 1 cesium 132.905 [Xe]6s¹	56 鋇 **Ba** 2 barium 137.327 [Xe]6s²	71 鑥 **Lu** 3 lutetium 174.97 [Xe]6s²4f¹⁴5d¹	72 鉿 **Hf** 4 hafnium 178.49 [Xe]6s²4f¹⁴5d²	73 鉭 **Ta** 5 tantalum 180.95 [Xe]6s²4f¹⁴5d³	74 鎢 **W** 6,5,4,3,2 tungsten 183.84 [Xe]6s²4f¹⁴5d⁴	75 錸 **Re** 7,6,4,2,-1 rhenium 186.21 [Xe]6s²4f¹⁴5d⁵	76 鋨 **Os** 2,3,4,6,8 osmium 190.23 [Xe]6s²4f¹⁴5d⁶	77 銥 **Ir** 2,3,4,6 iridium 192.22 [Xe]6s²4f¹⁴5d⁷	78 鉑 **Pt** 2,4 platinum 195.08 [Xe]6s¹4f¹⁴5d⁹	79 金 **Au** 3,1 gold 196.97 [Xe]6s¹4f¹⁴5d¹⁰	80 汞 **Hg** 2,1 mercury 200.59 [Xe]6s²4f¹⁴5d¹⁰	81 鉈 **Tl** 3,1 thallium 204.38 [Xe]6s²4f¹⁴5d¹⁰6p¹	82 鉛 **Pb** 4,2 lead 207.2 [Xe]6s²4f¹⁴5d¹⁰6p²	83 鉍 **Bi** 3,5 bismuth 208.98 [Xe]6s²4f¹⁴5d¹⁰6p³	84 釙 **Po** 4,2 polonium [209] [Xe]6s²4f¹⁴5d¹⁰6p⁴	85 砹 **At** ±1,3,5,7 astatine [210] [Xe]6s²4f¹⁴5d¹⁰6p⁵	86 氡 **Rn** radon [222] [Xe]6s²4f¹⁴5d¹⁰6p⁶
87 鈁 **Fr** 1 francium [223] [Rn]7s¹	88 鐳 **Ra** 2 radium [226] [Rn]7s²	103 鐒 **Lr** 3 lawrencium [262] [Rn]7s²5f¹⁴6d¹	104 鑪 **Rf** rutherfordium [267] [Rn]7s²5f¹⁴6d²	105 鉿 **Db** dubnium [268] [Rn]7s²5f¹⁴6d³	106 𨧀 **Sg** seaborgium [271] [Rn]7s²5f¹⁴6d⁴	107 鈨 **Bh** bohrium [270] [Rn]7s²5f¹⁴6d⁵	108 鏞 **Hs** hassium [277] [Rn]7s²5f¹⁴6d⁶	109 鏌 **Mt** meitnerium [276] [Rn]7s²5f¹⁴6d⁷	110 𫟼 **Ds** darmstadtium [281] [Rn]7s¹5f¹⁴6d⁹	111 錀 **Rg** roentgenium [282] [Rn]7s¹5f¹⁴6d¹⁰	112 鿔 **Cn** copernicium [285] [Rn]7s²5f¹⁴6d¹⁰	113 鉨 **Nh** nihonium [285] [Rn]7s²5f¹⁴6d¹⁰7p¹	114 鈇 **Fl** flerovium [289] [Rn]7s²5f¹⁴6d¹⁰7p²	115 鏌 **Mc** moscovium [289] [Rn]7s²5f¹⁴6d¹⁰7p³	116 鉝 **Lv** livermorium [293] [Rn]7s²5f¹⁴6d¹⁰7p⁴	117 𥓓 **Ts** tennessine [294] [Rn]7s²5f¹⁴6d¹⁰7p⁵	118 奧 **Og** oganesson [294] [Rn]7s²5f¹⁴6d¹⁰7p⁶

57-70 **lanthanoids** **鑭系元素**	57 鑭 **La** 3 lanthanum 138.91 [Xe]6s²5d¹	58 鈰 **Ce** 3,4 cerium 140.12 [Xe]6s²4f¹5d¹	59 鐠 **Pr** 3,4 praseodymium 140.91 [Xe]6s²4f³	60 釹 **Nd** 3 neodymium 144.24 [Xe]6s²4f⁴	61 鉕 **Pm** 3 promethium [145] [Xe]6s²4f⁵	62 釤 **Sm** 3,2 samarium 150.36 [Xe]6s²4f⁶	63 銪 **Eu** 3,2 europium 151.96 [Xe]6s²4f⁷	64 釓 **Gd** 3 gadolinium 157.25 [Xe]6s²4f⁷5d¹	65 鋱 **Tb** 3,4 terbium 158.93 [Xe]6s²4f⁹	66 鏑 **Dy** 3 dysprosium 162.50 [Xe]6s²4f¹⁰	67 鈥 **Ho** 3 holmium 164.93 [Xe]6s²4f¹¹	68 鉺 **Er** 3 erbium 167.26 [Xe]6s²4f¹²	69 銩 **Tm** 3,2 thulium 168.93 [Xe]6s²4f¹³	70 鐿 **Yb** 3,2 ytterbium 173.04 [Xe]6s²4f¹⁴
actinoids **錒系元素**	89 錒 **Ac** 3 actinium [227] [Rn]7s²6d¹	90 釷 **Th** 4 thorium 232.04 [Rn]7s²6d²	91 鏷 **Pa** 5,4 protactinium 231.04 [Rn]7s²5f²6d¹	92 鈾 **U** 6,5,4,3 uranium 238.03 [Rn]7s²5f³6d¹	93 鎿 **Np** 6,5,4,3 neptunium [237] [Rn]7s²5f⁴6d¹	94 鈽 **Pu** 6,5,4,3 plutonium [244] [Rn]7s²5f⁶	95 鎇 **Am** 6,5,4,3 americium [243] [Rn]7s²5f⁷	96 鋦 **Cm** 3 curium [247] [Rn]7s²5f⁷6d¹	97 錕 **Bk** 4,3 berkelium [247] [Rn]7s²5f⁹	98 鐨 **Cf** 3 californium [251] [Rn]7s²5f¹⁰	99 鑀 **Es** 3 einsteinium [252] [Rn]7s²5f¹¹	100 鐨 **Fm** 3 fermium [257] [Rn]7s²5f¹²	101 鍆 **Md** 3,2 mendelevium [258] [Rn]7s²5f¹³	102 鍩 **No** 3,2 nobelium [259] [Rn]7s²5f¹⁴

atomic num 元素
symbol oxidation states
element name
atomic weight
electron configuration

alkalai metals	alkaline earths	lanthanoid	actinoid	transition metals
other metals	semi-metals	non-metals	halogens	noble gases

Solid **Liquid** **Gas** **Synthetic** **Unknown**

English-Chinese Periodic Table of Elements 英漢元素周期表

1 / Ia																	18 / VIIIa
1 氫 **H** 1 hydrogen 1.0079 1s¹																	2 氦 **He** helium 4.0026 1s²
3 鋰 **Li** 1 lithium 6.941 [He]2s¹	4 鈹 **Be** 2 beryllium 9.0122 [He]2s²											5 硼 **B** 3 boron 10.811 [He]2s²2p¹	6 碳 **C** 4,2 carbon 12.011 [He]2s²2p²	7 氮 **N** ±3,5,4,2 nitrogen 14.007 [He]2s²2p³	8 氧 **O** -2 oxygen 15.999 [He]2s²2p⁴	9 氟 **F** -1 fluorine 18.998 [He]2s²2p⁵	10 氖 **Ne** neon 20.180 [He]2s²2p⁶
11 鈉 **Na** 1 sodium 22.990 [Ne]3s¹	12 鎂 **Mg** 2 magnesium 24.305 [Ne]3s²											13 鋁 **Al** 3 aluminum 26.982 [Ne]3s²3p¹	14 硅 **Si** 4 silicon 28.086 [Ne]3s²3p²	15 磷 **P** ±3,5,4 phosphorus 30.974 [Ne]3s²3p³	16 硫 **S** ±2,4,6 sulfur 32.065 [Ne]3s²3p⁴	17 氯 **Cl** ±1,3,5,7 chlorine 35.453 [Ne]3s²3p⁵	18 氬 **Ar** argon 39.948 [Ne]3s²3p⁶
19 鉀 **K** 1 potassium 39.098 [Ar]4s¹	20 鈣 **Ca** 2 calcium 40.078 [Ar]4s²	21 鈧 **Sc** 3 scandium 44.956 [Ar]4s²3d¹	22 鈦 **Ti** 4 titanium 47.867 [Ar]4s²3d²	23 釩 **V** 5,3 vanadium 50.942 [Ar]4s²3d³	24 鉻 **Cr** 6,3,2 chromium 51.996 [Ar]4s¹3d⁵	25 錳 **Mn** 7,6,4,2,3 manganese 54.938 [Ar]4s²3d⁵	26 鐵 **Fe** 2,3 iron 55.845 [Ar]4s²3d⁶	27 鈷 **Co** 2,3 cobalt 58.933 [Ar]4s²3d⁷	28 鎳 **Ni** 2,3 nickel 58.693 [Ar]4s²3d⁸	29 銅 **Cu** 2,1 copper 63.546 [Ar]4s¹3d¹⁰	30 鋅 **Zn** 2 zinc 65.409 [Ar]4s²3d¹⁰	31 鎵 **Ga** 3 gallium 69.723 [Ar]4s²3d¹⁰4p¹	32 鍺 **Ge** 4 germanium 72.64 [Ar]4s²3d¹⁰4p²	33 砷 **As** ±3,5 arsenic 74.922 [Ar]4s²3d¹⁰4p³	34 硒 **Se** ±2,4,6 selenium 78.96 [Ar]4s²3d¹⁰4p⁴	35 溴 **Br** ±1,5 bromine 79.904 [Ar]4s²3d¹⁰4p⁵	36 氪 **Kr** krypton 83.798 [Ar]4s²3d¹⁰4p⁶
37 銣 **Rb** 1 rubidium 85.468 [Kr]5s¹	38 鍶 **Sr** 2 strontium 87.62 [Kr]5s²	39 釔 **Y** 3 yttrium 88.906 [Kr]5s²4d¹	40 鋯 **Zr** 4 zirconium 91.224 [Kr]5s²4d²	41 鈮 **Nb** 5,3 niobium 92.906 [Kr]5s¹4d⁴	42 鉬 **Mo** 6,5,4,3,2 molybdenum 95.94 [Kr]5s¹4d⁵	43 鎝 **Tc** 7 technetium [98] [Kr]5s²4d⁵	44 釕 **Ru** 2,3,4,6,8 ruthenium 101.07 [Kr]5s¹4d⁷	45 銠 **Rh** 2,3,4 rhodium 102.91 [Kr]5s¹4d⁸	46 鈀 **Pd** 2,4 palladium 106.42 [Kr]4d¹⁰	47 銀 **Ag** 1 silver 107.87 [Kr]5s¹4d¹⁰	48 鎘 **Cd** 2 cadmium 112.41 [Kr]5s²4d¹⁰	49 銦 **In** 3 indium 114.82 [Kr]5s²4d¹⁰5p¹	50 錫 **Sn** 4,2 tin 118.71 [Kr]5s²4d¹⁰5p²	51 銻 **Sb** ±3,5 antimony 121.76 [Kr]5s²4d¹⁰5p³	52 碲 **Te** ±2,4,6 tellurium 127.60 [Kr]5s²4d¹⁰5p⁴	53 碘 **I** ±1,5,7 iodine 126.90 [Kr]5s²4d¹⁰5p⁵	54 氙 **Xe** xenon 131.29 [Kr]5s²4d¹⁰5p⁶
55 銫 **Cs** 1 cesium 132.905 [Xe]6s¹	56 鋇 **Ba** 2 barium 137.327 [Xe]6s²	71 鑥 **Lu** 3 lutetium 174.97 [Xe]6s²4f¹⁴5d¹	72 鉿 **Hf** 4 hafnium 178.49 [Xe]6s²4f¹⁴5d²	73 鉭 **Ta** 5 tantalum 180.95 [Xe]6s²4f¹⁴5d³	74 鎢 **W** 6,5,4,3,2 tungsten 183.84 [Xe]6s²4f¹⁴5d⁴	75 錸 **Re** 7,6,4,2,-1 rhenium 186.21 [Xe]6s²4f¹⁴5d⁵	76 鋨 **Os** 2,3,4,6,8 osmium 190.23 [Xe]6s²4f¹⁴5d⁶	77 銥 **Ir** 2,3,4,6 iridium 192.22 [Xe]6s²4f¹⁴5d⁷	78 鉑 **Pt** 2,4 platinum 195.08 [Xe]6s¹4f¹⁴5d⁹	79 金 **Au** 3,1 gold 196.97 [Xe]6s¹4f¹⁴5d¹⁰	80 汞 **Hg** 2,1 mercury 200.59 [Xe]6s²4f¹⁴5d¹⁰	81 鉈 **Tl** 3,1 thallium 204.38 [Xe]6s²4f¹⁴5d¹⁰6p¹	82 鉛 **Pb** 4,2 lead 207.2 [Xe]6s²4f¹⁴5d¹⁰6p²	83 鉍 **Bi** 3,5 bismuth 208.98 [Xe]6s²4f¹⁴5d¹⁰6p³	84 釙 **Po** 4,2 polonium [209] [Xe]6s²4f¹⁴5d¹⁰6p⁴	85 砹 **At** ±1,3,5,7 astatine [210] [Xe]6s²4f¹⁴5d¹⁰6p⁵	86 氡 **Rn** radon [222] [Xe]6s²4f¹⁴5d¹⁰6p⁶
87 鈁 **Fr** 1 francium [223] [Rn]7s¹	88 鐳 **Ra** 2 radium [226] [Rn]7s²	103 鐒 **Lr** 3 lawrencium [262] [Rn]7s²5f¹⁴6d¹	104 𨧀 **Rf** rutherfordium [267] [Rn]7s²5f¹⁴6d²	105 𨧄 **Db** dubnium [268] [Rn]7s²5f¹⁴6d³	106 𨨏 **Sg** seaborgium [271] [Rn]7s²5f¹⁴6d⁴	107 𨭆 **Bh** bohrium [270] [Rn]7s²5f¹⁴6d⁵	108 𨨫 **Hs** hassium [277] [Rn]7s²5f¹⁴6d⁶	109 鿏 **Mt** meitnerium [276] [Rn]7s²5f¹⁴6d⁷	110 𫟼 **Ds** darmstadtium [281] [Rn]7s¹5f¹⁴6d⁹	111 錀 **Rg** roentgenium [282] [Rn]7s¹5f¹⁴6d¹⁰	112 鿔 **Cn** copernicium [285] [Rn]7s²5f¹⁴6d¹⁰	113 鉨 **Nh** nihonium [285] [Rn]7s²5f¹⁴6d¹⁰7p¹	114 鈇 **Fl** flerovium [289] [Rn]7s²5f¹⁴6d¹⁰7p²	115 鏌 **Mc** moscovium [289] [Rn]7s²5f¹⁴6d¹⁰7p³	116 鉝 **Lv** livermorium [293] [Rn]7s²5f¹⁴6d¹⁰7p⁴	117 𨧍 **Ts** tennessine [294] [Rn]7s²5f¹⁴6d¹⁰7p⁵	118 鿫 **Og** oganesson [294] [Rn]7s²5f¹⁴6d¹⁰7p⁶

alkalai metals	alkaline earths	lanthanoid	actinoid	transition metals
other metals	semi-metals	non-metals	halogens	noble gases

atomic num 元素
symbol oxidation states
element name
atomic weight
electron configuration

Solid **Liquid** **Gas** **Synthetic** **Unknown**

57-70 **lanthanoids** 鑭系元素

57 鑭 **La** 3 lanthanum 138.91 [Xe]6s²5d¹	58 鈰 **Ce** 3,4 cerium 140.12 [Xe]6s²4f¹5d¹	59 鐠 **Pr** 3,4 praseodymium 140.91 [Xe]6s²4f³	60 釹 **Nd** 3 neodymium 144.24 [Xe]6s²4f⁴	61 鉕 **Pm** 3 promethium [145] [Xe]6s²4f⁵	62 釤 **Sm** 3,2 samarium 150.36 [Xe]6s²4f⁶	63 銪 **Eu** 3,2 europium 151.96 [Xe]6s²4f⁷	64 釓 **Gd** 3 gadolinium 157.25 [Xe]6s²4f⁷5d¹	65 鋱 **Tb** 3,4 terbium 158.93 [Xe]6s²4f⁹	66 鏑 **Dy** 3 dysprosium 162.50 [Xe]6s²4f¹⁰	67 鈥 **Ho** 3 holmium 164.93 [Xe]6s²4f¹¹	68 鉺 **Er** 3 erbium 167.26 [Xe]6s²4f¹²	69 銩 **Tm** 3,2 thulium 168.93 [Xe]6s²4f¹³	70 鐿 **Yb** 3,2 ytterbium 173.04 [Xe]6s²4f¹⁴

actinoids 錒系元素

89 錒 **Ac** 3 actinium [227] [Rn]7s²6d¹	90 釷 **Th** 4 thorium 232.04 [Rn]7s²6d²	91 鏷 **Pa** 5,4 protactinium 231.04 [Rn]7s²5f²6d¹	92 鈾 **U** 6,5,4,3 uranium 238.03 [Rn]7s²5f³6d¹	93 鎿 **Np** 6,5,4,3 neptunium [237] [Rn]7s²5f⁴6d¹	94 鈽 **Pu** 6,5,4,3 plutonium [244] [Rn]7s²5f⁶	95 鎇 **Am** 6,5,4,3 americium [243] [Rn]7s²5f⁷	96 鋦 **Cm** 3 curium [247] [Rn]7s²5f⁷6d¹	97 錕 **Bk** 4,3 berkelium [247] [Rn]7s²5f⁹	98 鐨 **Cf** 3 californium [251] [Rn]7s²5f¹⁰	99 鑀 **Es** 3 einsteinium [252] [Rn]7s²5f¹¹	100 鐨 **Fm** 3 fermium [257] [Rn]7s²5f¹²	101 鍆 **Md** 3,2 mendelevium [258] [Rn]7s²5f¹³	102 諾 **No** 3,2 nobelium [259] [Rn]7s²5f¹⁴

Reference

[1] T. Mass. (Mar. 1, 2018). Tex community, [Online]. Available: https://tex.stackexchange.com.

[2] Anonymous. (Jan. 1, 2018). Latex tutorial, [Online]. Available: https://www.latex-tutorial.com/tutorials/.

[3] 西二闲画生. (Feb. 1, 2018). Chinese history and politics blog, [Online]. Available: https://blog.zhstar.win.

Appendices

Appendix A

Too Big to Fit

A.1 Appendix Tips

The *appendix* package provides *appendices* environment that renames each chapter to Appendix A, Appendix B, etc. Similarly, sections are renamed to A.1, A.2, etc. respectively.

A.2 Embedded fonts in PDF

Here is an example of PDF file generated by LaTeX. We use command tool *pdffonts* to examine embedded fonts:

```
name                                 type              encoding         emb sub uni object ID
------------------------------------ ----------------- ---------------- --- --- --- ---------
TVICFK+Tinos                         CID TrueType      Identity-H       yes yes yes      5  0
XYTJRZ+AdobeSongStd-Light-Identity-H CID Type 0C       Identity-H       yes yes no       7  0
NJETOY+Tinos-Italic                  CID TrueType      Identity-H       yes yes yes      9  0
VMQFJA+Tinos-Bold                    CID TrueType      Identity-H       yes yes yes     15  0
ILVYSG+NotoSansHans-Bold-Identity-H  CID Type 0C       Identity-H       yes yes yes     20  0
HEDEEX+migu-1m-regular               CID TrueType      Identity-H       yes yes yes     50  0
KQTNVZ+CMSY10                        Type 1C           Builtin          yes yes no      55  0
```

Listing A.1: LaTeX 内嵌字体

A.3 pgfplotstable template

```
\begin{table}[tbp]
  \centering{} \pgfplotstabletypeset[ multicolumn names, % allows
  to have column header name col sep=comma, % the seperator in our
  .csv file
  display columns/0/.style={ % numbering starts at 0
    column name=$Ampere$, % header name of first column
    column type={S},string type % use siunitx for formatting
  }, display columns/1/.style={ column name=$Voltage$, column
    type={S},string type }, display columns/2/.style={ column
    name=$Energy$, column type={S},string type }, every head
  row/.style={ before row={\toprule}, % have a rule at top
    after row={ \si{\ampere} & \si{\volt} & \si{joule} \\ % the
      siunitx units seperated by & \midrule % rule under units
    } }, every last row/.style={ after row=\bottomrule % rule at
    bottom }, ]{pgfplotstable.csv} % filename/path to file
  \caption{Table automation from .csv file.}
  \label{table-automation-from-csv}
\end{table}
```

Listing A.2: pgfplotstable template

```
\begin{figure}[!h]
  \centering
  \begin{tikzpicture}
    \begin{axis}[
      width = \linewidth, % Scale the plot to \linewidth
      grid = major, grid style = dashed,
      xlabel = Voltage $U$, ylabel = Currency $I$, % Set the labels
      x unit = \si{\volt}, y unit = \si{\ampere}, % Set the respective units
      % axis lines = left % only display the left and bottom axes
      legend style = { at = {(0.5,-0.2)}, anchor = north }, % Put
      the legend below the plot x tick label style = { rotate =
        90, anchor = east } %
      Display labels sideways ]
      % add a plot from table; you select the columns by using the
      % actual column header name in the .csv file
      \addplot table[x=value 1,y=value 2,col
      sep=comma]{pgfplots.csv}; \legend{$U$ - $I$}
      % add another plot
      \addplot {x^2 - 2*x + 1}; % add a tailing semicolon
      \addlegendentry{$x^2 - 2x + 1$} % use addlegendentry instead
      of legend
    \end{axis}
  \end{tikzpicture}
  \caption{pgfplots by table csv file}
  \label{fig:pgfplots-by-table-csv-file}
\end{figure}
```

Listing A.3: pgfplots template

A.4 missTime

```
#!/bin/bash

if [[ "$1" = @(-h|--help) ]]
then
    printf 'Usage: bash missTime [<refr-Yy/Nn> [<number-of-test> [<input-file>]]]\n
refr-Yy/Nn:\t\trefresh or not (default: Y).
number-of-test:\t\tnumber of tests (default: 5).
input-file:\t\tfile contains the log.\n
bash missTime y 2 input.raw
cclog hpc access 5m | grep -E "TCP_HIT.*iosapps.itunes.apple.com.*headers=" | tail -2
       | bash missTime\n'

    exit 0
fi

_refr_opt=${1:-Y}
if [[ ! "$_refr_opt" =~ ^[YyNn]$ ]]
then
    printf '%s: Yy/Nn expected!\n' "$1"
    exit 1
fi

_num_of_tests=${2:-5}
if [[ ! "$_num_of_tests" =~ ^[0-9]*$ ]]
then
    printf '%s: integer expected!\n' "$_num_of_tests"
fi

_infd=0; [[ -f "$3" ]] && exec {_infd}< "$3"

read -r _ _ _ _ip _ < <(ip -4 -o addr show scope global dev bond0) ; _ip="${_ip%/*}"

_headers_regex=$'\^\~\$headers=\'(.*)\)@\|#\(\'' #'
_https_regex='^https://'
_refr_port="770"

_log_dir="${HOME}/logs-missTime"
mkdir -p "${_log_dir}"
for f in *.log; do mv "$f" "${_log_dir}/" 2>/dev/null; done

while IFS=$' \t\n' read -p "Log expected:" -u "${_infd}" -r line
do
    mapfile -t _log_array < <( awk -F'[[:space:]]*\)@(in|out)#\([[:space:]]*' '{ for
        (i = 1; i <= NF; ++i) print $i; }' <<< "${line}" )

    _std_log="${_log_array[0]}" _inr_log="${_log_array[1]}"
    [[ "${_log_array[2]}" ]] && _ext_log="${_log_array[2]}"

    read -ra _std_array <<< "${_std_log}" ; _log_time="${_std_array[0]}" _url="${
       _std_array[6]}"

    if [[ ! "${_inr_log}" =~ $_headers_regex ]]
    then
  printf '%s: log format error.\n' "${_log_time}"
  continue
    fi
    tmp_headers="${BASH_REMATCH[1]})@|#(" _headers_array=()
    while [[ $tmp_headers ]]
    do
  _headers_array+=( "${tmp_headers%%\)\@\|\#\(*}" )
  tmp_headers="${tmp_headers#*\)\@\|\#\(}"
    done; unset tmp_headers

    IFS='/:' read -r _scheme _ _ _host _uri <<< "${_url}"
    if [[ "${_scheme}" == "https" ]]
    then
  _protocol_port=443
    else
  _protocol_port=80
    fi
    _resolve="${_host}:${_protocol_port}:${_ip}"

    for f in full real; do declare "_$f"="${_log_time}-${_num_of_tests}-$f.log"; done
    for f in "${_full}" "${_real}"; do echo -n >| "$f"; done
    {
  printf -- '########## missTime ##########\n'
  printf 'HN:\t%s\nIP:\t%s\nLOG:\t%s\nRefr:\t%s\nTest:\t%s\n\n' "$(hostname)" "${_ip}"
       "${_log_time}" "${_refr_opt}" "${_num_of_tests}"
  printf 'cclog:\n%s\n\n' "${line}"
  printf 'standard part:\n%s\n\ninternal part:\n%s\n\n' "${_std_log}" "${_inr_log}"
  [[ "${_ext_log}" ]] && printf 'external part:\n%s\n\n' "${_ext_log}"
  printf 'curl -ksSI \\\n' ; printf -- $'-H \'%s\' \\\n' "${_headers_array[@]}" ;
      printf -- $'-H \'x-c3-debug:enabled\' \\\n--resolve \'%s\' \\\n\'%s\'\n\n' "${
      _resolve}" "${_url}" #'
  printf 'curl -ksSI ' ; printf -- '-H %q ' "${_headers_array[@]}" ; printf -- '-H %q
      --resolve %q %q\n\n' "x-c3-debug:enabled" "${_resolve}" "${_url}"
    } | tee -a "${_full}"

    _headers_array=( "${_headers_array[@]/#/-H}" )
    printf '' >| "curl-reply.log"
    ( set -x; { curl -ksSI "${_headers_array[@]}" -H "x-c3-debug:enabled" --resolve "$
       {_resolve}" "${_url}" ; } 2>&1 ) | tee -a "${_full}" "curl-reply.log"
```

Listing A.4: missTime Line by Line

```
    [[ "$_refr_opt" == [Nn] ]] && continue

    declare -A _reply_array=()
    while IFS=' :' read -r key value
    do
  [[ "$key" && "$key" != [[:space:]] && "$key" != + ]] || continue
  _reply_array[${key}]=${value%$'\r'}
    done < "curl-reply.log"

    {
  # for key in "${!_reply_array[@]}"; do printf '%s: %s\n' "$key" "${_reply_array[$key
      ]}" ; done ; printf '\n'
  printf 'X-Cache: %s\n' "${_reply_array[X-Cache]}" ; printf 'X-Cache-Remote: %s\n' "$
      {_reply_array[X-Cache-Remote]}" ; printf 'CC_CACHE: %s\n\n' "${_reply_array[
      CC_CACHE]}"
    } | tee -a "${_full}"

    shopt -q extglob; _extglob_set=$? ; (( _extglob_set )) && shopt -s extglob
    _delete_url=${_reply_array[X-True-Cache-Key]#http?(s)://}
    (( _extglob_set )) && shopt -u extglob

    [[ "${_scheme}" == "https" ]] && _delete_url=${_delete_url/\//:443/}
    _refr_url=http://127.0.0.1:${_refr_port}/delete/${_delete_url}
    printf 'refr url: %s\n' "${_refr_url}" | tee -a "${_full}"

    for i in $(seq 1 ${_num_of_tests})
    do
  printf -- '---%sth---\n' "${i}" | tee -a "${_full}" "${_real}"

  printf 'refr code: ' | tee -a "${_full}"
  curl -ksSo /dev/null -w "%{response_code}" -H "User-Agent: DataDelete" "${_refr_url}
      " | tee -a "${_full}"
  printf '\n' | tee -a "${_full}"

  sleep 0.05s
  { time curl -kLsSvo /dev/null "${_headers_array[@]}" -H "x-c3-debug:enabled" --
      resolve "${_resolve}" "${_url}" 2>&1 | awk 'BEGIN {IGNORECASE=1}; /<[[:space
      :]].*[-_]cache:/ {print substr($0, 3) }' ; } 2>&1 | tee -a "${_full}"

  ed -s "${_full}" <<< $'$-3d\nw'
  ed -s "${_full}" <<< $'$-3,-2p' >> "${_real}"
    done
    printf '%s\n' '1d' w | ed -s "$_full" # sed -i -e '1d' "$_full"

done

rm -f "curl-reply.log"
```

Listing A.5: missTime Line by Line

List of Figures

List of Tables

Listings

Postscript

This is the summary note pertaining to the book. This is the summary note pertaining to the book. This is the summary note pertaining to the book. This is the summary note pertaining to the book.