目录

1 TEX 中基本概念 **2**

1.1 命令 (宏) 2

1.2 常用宏 2

1.2.1 长度类 2

1.2.2 catcode 类 2

1.2.3 盒子类 3

1.2.4 glue 类 3

1.3 变量 3

1.4 catcode 3

2 TEX 编程 **5**

2.1 变量定义 5

2.2 函数定义 6

2.2.1 格式控制函数 6

2.2.2 数值计算函数 6

2.3 流程控制 7

2.3.1 条件判断函数 7

2.3.2 流程控制语句 8

2.4 循环 9

2.4.1 小型循环 9

2.4.2 大型循环 9

2.4.3 死循环 10

2.4.4 递归 10

2.5 数据结构 12

2.5.1 结构体 12

2.5.2 链表 12

1 TEX 中基本概念

1.1 命令 (宏)

对于想要自定义命令的选手, 但是没有自定义的思路的情况下, 你们可以直接查看 LATEX(TEX) 的内部命令定义, 在命令行下使用如下的命令 latexdef(texdef), 然后你就可以得到如下的输出:

```
\LaTeX:
macro:->\protect \LaTeX

\LaTeX :
\long macro:->L\kern -.36em{\sbox \z@ T\vbox to\ht \z@ {\hbox{\check@mathfonts \fontsize \sf@size \z@ \math@fontsfalse \selectfont A}\vss }}\kern -.15em\TeX
```

然后你就可以仿照着官方源码里卖的定义自己弄了, 尽管你可能看不懂但是这不重要, 慢慢模仿就会了.

1.2 常用宏

在这里推荐一些有用的宏, 或者说是你想要实现某些功能时已经定义好的宏.[1]

1.2.1 长度类

- `\linewidth` → 计算当前行的剩余长度
- `\dimexpr` → 用于长度的计算, 比如`\linewidth-#1`

1.2.2 catcode 类

- `\catcode` → 改变字符对应的 catcode 编码, 属于 TEX 的黑魔法
- `\makeatletter` → 把 @ 的 catcode 从 12 变成 11, 因为自定义宏中的字符的 catcode 值必须为 11
- `\makeatother` → 把 @ 的 catcode 从 11 变成 12
- `\string` → 把命令后的第一个 `token` 原样输出, 可以看作一个命令式声明
- `\detokenize` → 把 `{}` 中的全部内容原样输出

[1] 具体的说明可以参见 TEX For Impatiant

1.2.3 盒子类

- \leavevmode → 离开竖直模式 (比如每一段的开头), 产生一个长度可忽略的空格, 进入水平模式
- \hbox → 产生一个水平盒子,\hbox to 100pt{**}, 经常和\kern 命令联合使用
- \vbox → 和\hbox 类似, 产生一个竖直排列的盒子
- \vrule → \vrule height4pt width3pt depth2pt, 一个实心盒子,\vrule 同理
- \strut → 用于插入一个高度和深度都为 0pt 的不可见盒子, 确保当前行的高度和深度至少与一个普通大小的大写字母相同, 以获得更好的垂直对齐效果
- \moveright → \moveright 1in \vbox {**}, 把 vbox 向右移动 1 in,\moveleft 同理

1.2.4 glue 类

- \kern → 调整水平间距,\kern1em; 正表示向右, 负表示向左
- \raise → 调整竖直间距,\raise.4ex; 正表示向上, 负表示向下
- \lower → 调整竖直间距,\lower-10.5pt; 正表示向下, 负表示向上
- \vphantom → 用于创建一个垂直尺寸与参数相同的不可见盒子, 调整上下间距.
- \mathstrut → 用于调整数学公式的垂直大小, 以保持一致的行间距和上下位置的对齐, \sum _{\mathstrut i=1}^{\mathstrut n} x_i.

  (调整结果)$\sum_{i=1}^{n} x_i$ → (原始结果)$\sum_{i=1}^{n} x_i$

1.3 变量

其实就是采用类似的宏定义的方法, 其实就是常用的\def, \newcommand

1.4 catcode

为了进一步使用, 你需要熟悉 catcode 的相关操作, 而且在 \catcode 命令中, 类别码 (category code) 后面的字符必须用反引号 ` 包围起来[2]. 这是因为反引号 ` 是 LaTeX

[2] 类别码后面的字符和反引号 ` 之间不应有空格, 否则空格也会被视为类别码的一部分, 可能引起错误

中的一种特殊字符, 用于指示后面的字符是一个单个字符而不是控制序列, 使用的语法就像是这样的:`\catcode`<token>=<num>`, 一个应用的例子就比如下面这段示例代码:

```
\catcode`0=0
0LaTeX
```

运行结果

0LaTeX

如果你想要把 0 对应的 catcode 改回来, 使用语句[3] :`\catcode`\0=12`. 下面我们就定义一个很常用的转义\ 的命令. 相信很多的朋友都想搞这个东西, 特别是在进行代码抄录时.

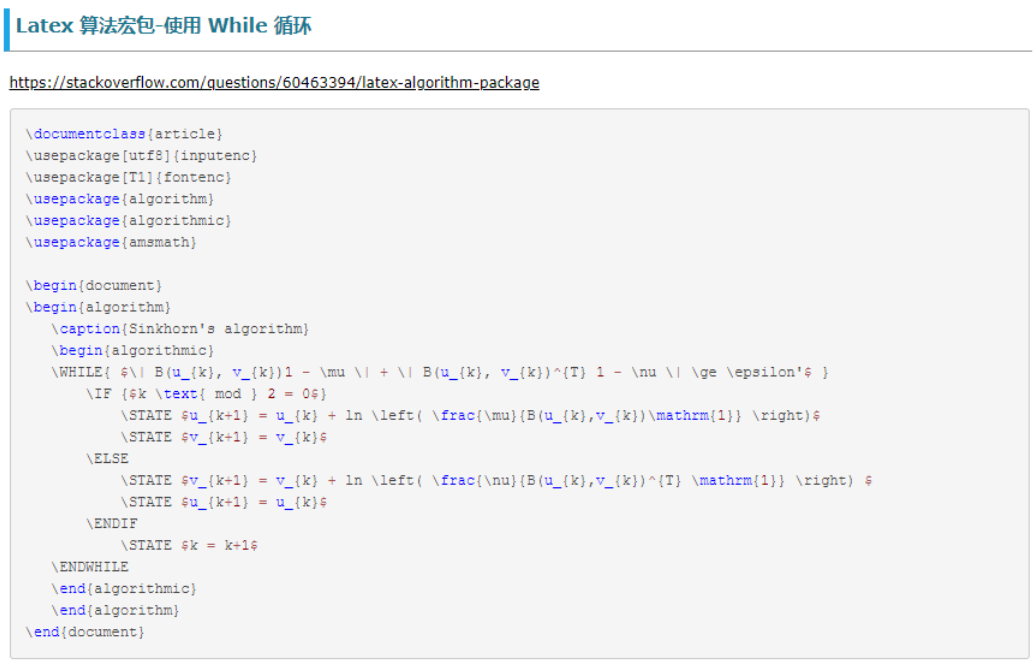
但通过在自定义的宏中执行修改 catcode 的命令不行, 我们可以使用`\string` 命令, 或者是处理多个命令的`\detokenize`.

[3] 注意: 不是`\catcode`0=12`, 主要就是应为此时的 0 已经是一个特殊字符了, 你需要转义

2 TEX 编程

注意: 我这里的编写程序是在不使用 LATEX3 的前提, 而仅仅只是使用 LATEX2ε 或者是 plain TEX 提供的命令.

而且很多的情况下你用浏览器去搜索 LATEX 编程时, 你往往只能得到下面的所搜结果:


Latex 算法宏包-使用 While 循环

https://stackoverflow.com/questions/60463394/latex-algorithm-package

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{algorithm}
\usepackage{algorithmic}
\usepackage{amsmath}

\begin{document}
\begin{algorithm}
   \caption{Sinkhorn's algorithm}
   \begin{algorithmic}
   \WHILE{ $\| B(u_{k}, v_{k})1 - \mu \| + \| B(u_{k}, v_{k})^{T} 1 - \nu \| \ge \epsilon'$ }
       \IF {$k \text{ mod } 2 = 0$}
           \STATE $u_{k+1} = u_{k} + ln \left( \frac{\mu}{B(u_{k},v_{k})\mathrm{1}} \right)$
           \STATE $v_{k+1} = v_{k}$
       \ELSE
           \STATE $v_{k+1} = v_{k} + ln \left( \frac{\nu}{B(u_{k},v_{k})^{T} \mathrm{1}} \right) $
           \STATE $u_{k+1} = u_{k}$
       \ENDIF
           \STATE $k = k+1$
   \ENDWHILE
   \end{algorithmic}
   \end{algorithm}
\end{document}
```

图 1: TEX 编程搜索结果

这个就是教你怎么写伪代码的帖子, 并不是教你像 C 那样使用 TEX 进行宏编程[4]的教程. 我当初就是没能找到合适的帖子[5], 可能也是因为我不会使用 Chrome 吧. 所以才用了今天的这篇文章, 并不是那么系统的介绍**宏编程**的文章.

其实你也可以认为我的这篇文章就是一个为了类比主流语法的**缝合怪**, 我并不会说什么. 因为我是按照目前主流的 C, C++, Python 的**编程范式**来组织这篇文章的, 并没有按照其他的例如 HasKell, Mathematica 这种函数式编程范式风格组织的.

2.1 变量定义

其实就是用一个符号去代表某个值, 我们可以使用如下的几个命令:

[4] 详细的, 系统的教程必然只能你自己去看 TeXByTopic, TheTexbook 这类的书学习. 学完后建议总结一下分享在博客, 扩大 TEX 的影响力, 促进知识平等

[5] 你可以在 LaTeXStudio 的官网中找到宏编程这个栏目, 但是由于我是白嫖党, 自然我会为了发扬白嫖精神, 写这篇教程 [doge]

- def: 定义
- gdef: 全局定义
- let: 命令的重命名复制
- newcommand: 最常用的定义方式
- NewDocumentCommand:xparse宏包提供

比如我使用 sex 表示性别, 定义一个变量[6]sex=female.

```
\def \sex {female}
\sex
```

运行结果 → female

2.2 函数定义

2.2.1 格式控制函数

类比 C 语言里面的函数, 无非就是输入一些参数, 然后给你返回一些参数, 其实还是用的上面的几个宏. 只不过我们这里的函数理解为**替换**好一点, 其实就是把一个命令替换为自己提前定义的内容, 只不过我们可以对**替换**的内容进行一些的格式设置. 下面我们定义一个用于把以文本复制一次的命令\copytext{text}, 它接受一个 text 文本, 然后把这个 text 复制一次输出, 把原本的 text 原样输出, 复制的 text 输出颜色为橙色.

```
\def\copytext#1{%
    #1\textcolor{orange}{#1}}
\copytext{Test}
```

运行结果 → TestTest

其实 LATEX 中环境的定义就是命令\def 的一种拓展而且, 无非就是在一个段落的前面和后面加了一个前置定义和后置定义, 这个没什么好说的.

2.2.2 数值计算函数

由于 TEX 这玩意儿的计算能力本来就不咋行, 在无数个文档中见过这个了 ([doge]). 所以如果你想要自定义一些进行数值运算的函数的话, 恐怕就得使用 xfp 这个宏包, 这个宏包的使用方法是十分的】简单的, 官方文档也就几页,10min 保证就会用了. 然后你就可以用它来定义一些内容了.

但是如果你不只是仅仅拘泥于 TEX 这个玩意儿用于计算的话, 那么你的选择就太多了, 你可以使用外部语言 Lua, Python, GNUPlot 的计算能力. 你甚至还可以使用 Mathematica, 要使用 Mathematica 你只需要安装一个 Mathematica 的云还是啥的, 在加上一个别人开发的 latexalpha2 宏包, 你就可以使用 Mathematica 的符号计算, 绘

[6] 注意: 和主流编程语言不同的是, 在 LATEX 中, 变量均为以\ 开头

图, 表格等几乎所有功能. 然而这一切的一切都只需要你在编译 TeX 源文件的时候开启--shell-escape 参数即可. 最后, 最麻烦的不过就是多编译几次的问题, 但是这个编译次数的问题也可以通过 latexmk 脚本完成.

最后在提一句, 其实在 tikz 绘图中, 我们很多时候都需要通过计算两个坐标, 从而得到目标坐标. 而 tikz 中使用的是 calu 库, 使用的语法就像是下下面这样的:

```
1 \draw[->] (0, 0)--($(a, b) + (c, d)$);
```

所以亦可以把 calu 库看作是定义好的函数库, 只不过我们的调用格式[7]不同罢了.

2.3 流程控制

2.3.1 条件判断函数

由于 LATEX 没有现代的**变量系统**的概念, 所以对于不同类型的变量我们需要使用不同的判断函数, 但是有一个好处, 它们都是类似的. 下面举例常用的判断函数:

- \ifx: 比较两个宏或控制序列是否具有相同的定义
- \ifmmode: 是否在数学模式 (环境)
- \ifvmode: 是否在竖直模式
- \ifhmode: 是否在水平模式
- \ifinner: 是否在内部模式, 内部模式包括数学模式和文本模式, 与竖直模式和水平模式不同
- \ifnum: 对两个数值进行比较,\countA<\countB
- \ifdim: 对两个长度进行比较,\ifdim\dimA<\dimB
- \ifodd: 检查一个数是否为奇数,\ifodd\countA
- \ifempty: 检查一个宏是否为空,\ifempty\foo

[7]我们自然也可以在 tikz 的坐标运算中使用 xfp 库

这些判函数可以与条件语句（如 \if、\else、\fi）结合使用, 用于根据条件执行不同的代码块. 下面以\ifodd 举例:

```
\def\numA{23}
% 前面一个 {} 用于说明语句结束
% 前一个 {} 也可以使用\relax 替代
\ifodd \numA
    {}\numA{}  Is Odd
\else
    {}\numA{}  Is Evan
\fi
```

运行结果

23 Is Odd

2.3.2 流程控制语句

需要注意的是 LaTeX 中是没有类似于 C 语言里面的逻辑运算符号 ||，&，!，<=，>=, 所以想要进行这些操作就只能我们写一个多重判断了[8], 其实就是多个单一判断的嵌套 [9].

- 比如我们要判断 $2 < x < 5$, 那么我们就需要向下面这么写:

```
\ifnum \x>2 \ifnum \x<5
    % 符合2<x<5时进行的操作
\fi\fi
```

- 进一步, 为了使用下面的这个 $\leq$ 功能, 我们可以使用类似如下的代码实现

```
\def\numA{12}
\def\numB{15}

\ifnum\numA<\numB\relax
    \numA<\numB
\else
    \ifnum\numA=\numB\relax
        \numA=\numB
    \else
        \numA>\numB
    \fi
\fi
```

[8] 当然, 如果你使用 LaTeX3 的话, 语法会更加的简洁, 包括下面的循环也是一样的

[9] 在 LaTeX 中使用判断时: **每一层**判断只能有一个\else

实际效果演示: 12<15

2.4 循环

其实接触过 LATEX 的 TiKZ 选手应该都知道里面有一个\foreach 循环,、这个东西可以用来绘制一些比较复杂的图形. 那么假如不在绘图的 tikzpicture 环境中使用\foreach 循环呢? 可行吗? — **可行的**.

2.4.1 小型循环

下面就是一个简单的演示, 演示了怎么遍历一个有限的, 或者是说比较短的列表

```
\def\list{
    1, 2, 3, 4, 5, 6, 7, 8, 9
}
\foreach \num in \list{
    \num
    % 当num=5时换行
    \ifnum\num=5
        \par
    \fi
}
```

\num = 1 \num = 2 \num = 3 \num = 4 \num = 5
\num = 6 \num = 7 \num = 8 \num = 9

肯定有人想说, 你的这个列表这么的短, 有必要用循环吗? 的确, 假如一个列表很短, 是没有必要使用循环的. 但是我们假如想要进行比较长的列表的遍历, 或者说是很长的列表的遍历, 比如一个拥有 999 个元素的列表, 我们自然希望有类似 C 语言的那种简洁的写法 \for {i=1; i<=1000; i++}{...}

2.4.2 大型循环

其实 LATEX 给我们提供了一个语法糖来达到列表的构建目的.{1,2,...,100}, 这个语法其实就是表示我们要创建一个 $1, 2, 3, ..., 100$ 这样的从 1 到 100 的列表, 而不用自己一个一个的写出来.

下面我们把 $0\circ \sim 30^\circ$ 之间的所有正弦值统统计算出来.[10]这个时候你应该不会想不开自己去计算吧, 但是你其实可以在调用外部程序, 比如 GNUPlot 去计算, 然后插入. 亦或者是直接在类似于 Matlab 或者是 Python 中算出来, 保存为 `txt` 格式, 然后自己到 `tablegenrator` 生成表格 `TeX` 源码, 然后你把源码复制进来, 微调一下编译. 下面我们使用循环生成数据:

```
%\usepackage{xfp}
\foreach \Angle in {1,2,...,30}{
    $\sin(\Angle^\circ)$ = \fpeval{sin(deg(\Angle))}
}
```

$\sin(1^\circ) = 0.01745240643728352$ $\sin(2^\circ) = 0.03489949670250098$ $\sin(3^\circ) = 0.05233595624294385$
$\sin(4^\circ) = 0.06975647374412532$ $\sin(5^\circ) = 0.08715574274765819$ $\sin(6^\circ) = 0.1045284632676535$
$\sin(7^\circ) = 0.1218693434051475$ $\sin(8^\circ) = 0.1391731009600655$ $\sin(9^\circ) = 0.1564344650402309$
$\sin(10^\circ) = 0.1736481776669304$ $\sin(11^\circ) = 0.1908089953765449$ $\sin(12^\circ) = 0.2079116908177594$
$\sin(13^\circ) = 0.2249510543438651$ $\sin(14^\circ) = 0.2419218955996678$ $\sin(15^\circ) = 0.2588190451025208$
$\sin(16^\circ) = 0.2756373558169993$ $\sin(17^\circ) = 0.2923717047227368$ $\sin(18^\circ) = 0.3090169943749475$
$\sin(19^\circ) = 0.3255681544571567$ $\sin(20^\circ) = 0.3420201433256688$ $\sin(21^\circ) = 0.3583679495453004$
$\sin(22^\circ) = 0.3746065934159121$ $\sin(23^\circ) = 0.3907311284892738$ $\sin(24^\circ) = 0.4067366430758003$
$\sin(25^\circ) = 0.4226182617406995$ $\sin(26^\circ) = 0.4383711467890775$ $\sin(27^\circ) = 0.4539904997395469$
$\sin(28^\circ) = 0.4694715627858909$ $\sin(29^\circ) = 0.4848096202463371$ $\sin(30^\circ) = 0.5000000000000001$

2.4.3 死循环

目前能够实现死循环的方法, 很简单, 直接让 TEX 的模式切换混乱就行了**(千万不要执行此操作, 除非你明确你自己在干什么)**. 具体见下面的代码[11]

```
\def\par{}
x\vskip 1pt
```

2.4.4 递归

由于 LATEX 中没有类似与 C 中那样的 `while` 循环, 自然也就没有对应的那种 `break` 语句. 但是你可以在 LATEX 中使用**递归**. 对, 就是**递归**, 就是那个萌新最爱的递归.

[10]注意: 由于要计算这种复杂的函数值, 所以我使用了一个 `xfp` 宏包.
[11]代码来源:https://liam.page/2020/04/28/hhh-explodes-TeX/

我们定义了一个递归函数[12]\myLoop, 它接受一个参数 #1 表示循环的次数. 在函数体中, 我们首先进行条件判断, 如果 #1 大于 0, 则执行循环体代码, 并通过递归调用\myLoop\numexpr#1-1\relax 来实现循环. 递归调用中的参数是 #1 减去 1, 即实现了循环次数的递减. 然后我们调用\myLoop{5} 来执行循环, 输出了着 5 次循环中的迭代信息

```
\newcounter{myLoopCounter}
\newcommand{\myLoop}[1]{%
    \setcounter{myLoopCounter}{#1}%
    \loop
        \ifnum\value{myLoopCounter}>0
        % 循环体代码
        Loop iteration: \the\value{myLoopCounter}\par
        \addtocounter{myLoopCounter}{-1} % 计数器减1
    \repeat
}

\myLoop{5}
```

Loop iteration: 5
Loop iteration: 4
Loop iteration: 3
Loop iteration: 2
Loop iteration: 1

其实 LATEX3 里面的这个代码才算真正的递归（只是代码片段, 不完整）

```
\cs_set:Npn \fibon:n #1{
    \int_compare:nNnTF{#1}={0}{0}
    {
        \int_compare:nNnTF{#1}={1}{1}
        {
            \fp_eval:n{\fibon:n{#1-1}+\fibon:n{#1-2}}
        }
    }
}
```

[12]但是这里其实并不是严格的递归, 有一点 while ... break 的感觉

2.5 数据结构

2.5.1 结构体

其实我们可以仿照 C 语言中结构体的定义, 自己创建一个**结构体**对象, 然后给它关联一些的属性, 从而进一步实现 C 语言中的各种复杂的数据结构. 这里的难点在于你要怎么关联对应的属性

2.5.2 链表

只要结构体能够实现, 那么这个链表的实现也是比较简单的, 只不过要找到一个类似于指针的东西??