ماکرونویسی

وفا خلیقی
persian-tex@tug.org

۸ مهر ۱۳۹۲

چکیده

فهرست مطالب

۱ پردازشگر تک

تک فایل ورودی `tex`. را در چهار مرحله پردازش می‌کند. هر مرحله خروجی مرحله قبل را قبول می‌کند و ورودی مرحله بعد را تولید می‌کند. ورودی اولین مرحله فایل `tex`. است و خروجی آخرین مرحله فایل `pdf`. است. این چهار مرحله عبارتند از:

۱. **پردازشگر ورودی.** تک فایل ورودی `tex`. را می‌خواند و محتویات این فایل را به یک سری نشانه (نشانه‌های کاراکتری و نشانه‌های دستوری) تبدیل می‌کند.

۲. **پردازشگر گسترش.** تعدادی ولی نه همه نشانه‌هایی که در مرحله اول تولید شده‌اند (ماکروها، شرط‌ها، و تعدادی از دستورات بدوی تک) گسترش می‌یابند. گسترش، روندی است که طی آن تعدادی از نشانه‌ها با نشانه‌های دیگر (یا هیچ نشانه‌ای) جایگزین می‌شوند.

۳. **پردازشگر اجرایی.** دستورات غیرقابل گسترش، اجرا می‌شوند. واگذاری‌ها (از جمله تعریف ماکروها) و ساختن لیست‌های افقی، عمودی، و ریاضی در این مرحله اتفاق می‌افتد.

۴. **پردازشگر دیداری.** لیست‌های افقی به پاراگراف‌ها، لیست‌های عمودی به صفحه‌ها، و لیست‌های ریاضی به فرمول‌ها تبدیل می‌شوند. خروجی `pdf`. در این مرحله تولید می‌شود. کاربر دست‌رسی به الگوریتمی که در این مرحله وجود دارد، ندارد اما این الگوریتم با استفاده از تعدادی از پارامترها قابل کنترل است.

۱.۱ پردازشگر ورودی

پردازشگر ورودی محتویات فایل `tex`. را به یک سری نشانه (یک لیست نشانه) تبدیل می‌کند. نشانه‌ها به سه دسته تقسیم می‌شوند: نشانه‌های کاراکتری، نشانه‌های دستوری، و نشانه‌های پارامتری.

۱.۱.۱ کاراکترهای ورودی

برای یک متن ساده ورودی، کاراکترها به نشانه‌های کاراکتری تبدیل می‌شوند. تک می‌تواند تعدادی از کاراکترهای ورودی را نادیده بگیرد (چندین فاصله پشت سر هم در فایل ورودی معمولاً تنها معادل یک فاصله است). تک همچنین خودش می‌تواند تعدادی نشانه را که در فایل ورودی وجود ندارد، وارد کند (به عنوان نمونه، نشانه‌های فاصله در آخر سطر، یا نشانه `par\` بعد از یک سطر خالی).

کاراکترها به ۱۶ رده تقسیم می‌شوند. هر رده به کاراکترهایی که به آن رده تعلق دارد، می‌گوید که چه نقشی دارند. تنها کاراکترهایی که به دو رده تعلق دارند، حروف‌چینی می‌شوند. رده‌های دیگر کاراکترهایی مانند `}`، `{`، `&`، و `#` دارند. یک نشانه کاراکتری یک جفت عدد است: کد کاراکتر و کد رده. کد رده هر کاراکتری قابل تغییر است.

نحوه ساختن دستور به این شکل است که تک از وروی تعدادی از کاراکترها (که کاراکتر اول کاراکتر گریز، بصورت پیش‌فرض `\` است) را می‌گیرد و سپس آن‌ها را به یک نشانه واحد تبدیل می‌کند.

برخورد پردازشگر تک با کدهای رده به یک ماشین که بین سه حالت داخلی سویچ می‌کند، تشبیه می‌شود: **ج** خط جدید، **م** میانه خط، **ق** قلم‌انداختن فاصله‌ها.

۲.۱.۱ پردازش ورودی دو-مرحله‌ای

خود پردازشگر ورودی یک پردازشگر دو-مرحله‌ای است. کاربر ممکن است به خاطر محدودیت‌های ترمینال، ویرایشگر، یا حتی سیستم عامل نتواند تعدادی از کاراکترها را وارد کند. تک مکانیزمی در اختیار قرار می‌دهد تا کاربر با استفاده از دو کاراکتر توان بتواند تمام کاراکترها را وارد کند.

به عنوان مثال، +^^ با k جایگزین می‌شود چون که کدهای اسکی + و k به انداز ۶۴ با هم تفاوت دارند. از آنجایی که این جایگزینی قبل از شکل‌گرفتن نشانه‌ها صورت می‌گیرد، \vs^^+ip 12cm همان \vskip 12cm خواهد بود.

توجه داشته باشید که مرحله اول پردازش ورودی، تبدیل کاراکترها به کاراکترها بدون درنظر گرفتن کدهای رده است. کدهای رده در مرحله دوم پردازش ورودی، زمانی که کاراکترها با جفت کردن کد رده و کد کاراکتر، به نشانه‌های کاراکتری تبدیل می‌شوند، وارد بازی می‌شوند.

۲.۱ پردازشگر گسترش

پردازشگر گسترش یک سری از نشانه‌ها را قبول می‌کند و در صورت امکان آن‌ها را یکی یکی گسترش می‌دهد تا اینکه تنها نشانه‌های غیرقابل گسترش باقی می‌مانند. گسترش ماکرو بهترین مثال از این دسته است: اگر یک دستور نام یک ماکرو باشد، این دستور (به همراه نشانه‌های پارامتری) با متن تعریف ماکرو جایگزین می‌شود.

ورودی پردازشگر گسترش توسط پردازشگر ورودی تهیه می‌شود. وقتی که نشانه‌ها از مرحله اول پردازش وارد مرحله دوم پردازش می‌شوند، گسترش پیدا می‌کنند و در آخر یک سری نشانه‌های غیرقابل گسترش تولید می‌شود که به خورد پردازشگر اجرایی در مرحله سوم پردازش داده می‌شود.

هرچند، زمانی که یک \edef یا \write در حال پردازش است، پردازشگر گسترش دوباره وارد بازی می‌شود. لیست نشانه پارامتری این دستورات بطوری گسترش پیدا می‌کند که انگار این لیست‌ها به جای اینکه آرگومان یک دستور باشند، در سطح بالاتری بوده‌اند.

۱.۲.۱ روند گسترش

۱. ببین که نشانه قابل گسترش یافتن هست یا خیر.

۲. اگر نشانه قابل گسترش یافتن نیست، آن را به لیست نشانه در حال ساخت بسپار و نشانه بعدی را در نظر بگیر.

۳. اگر نشانه قابل گسترش یافتن هست، آن را با گسترشش جایگزین کن. برای ماکروهای بدون پارامتر و تعداد کمی از دستورات بدوی مانند \jobname این یک جایگزینی ساده است. هر چند تک معمولاً برای ساختن جایگزین نشانه فعلی، باید تعدادی از نشانه‌های آرگومانی را بخورد. به عنوان مثال، اگر نشانه یک ماکرو با پارامتر باشد، برای ساختن آرگومان‌های این پارامتر(ها)، باید نشانه‌های کافی خورده شود.

۴. به گسترش دادن، با شروع از اولین نشانه گسترش، ادامه بده.

اینکه یک نشانه قابل گسترش یافتن هست یا خیر، تصمیم ساده‌ای است. ماکروها و کاراکترهای فعال، شرط‌ها، و تعدادی از دستورات بدوی قابل گسترش یافتن هستند، اما سایر نشانه‌ها قابل گسترش یافتن نیستند. بنابراین پردازشگر گسترش، ماکروها را با گسترششان

جایگزین می‌کند، شرط‌ها را می‌سنجد و بخش‌های بی‌ربط آن‌ها را حذف می‌کند، ولی نشانه‌هایی مانند \vskip و نشانه‌های کاراکتری، از جمله کاراکترهایی مانند دلار و آکولادها (دوابروها) دست‌نخورده به مرحله بعدی پردازش سپرده می‌شوند.

۲.۲.۱ موارد خاص: \expandafter، \noexpand، و \the

\expandafter⟨نشانه۱⟩⟨نشانه۲⟩

با

⟨نشانه۱⟩⟨گسترش نشانه۲⟩

جایگزین می‌شود. دو استثنا وجود دارد:

۱. اگر نشانه فعلی دستور \noexpand باشد، نشانه بعدی غیرقابل گسترش یافتن در نظر گرفته می‌شود (مثل اینکه نشانه بعدی \relax بوده باشد) و به لیست نشانه در حال ساخت سپرده می‌شود.

برای مثال، در تعریف زیر:

```
\edef\a{\noexpand\b}
```

\noexpand\b در زمان تعریف گسترش پیدا می‌کند. گسترش \noexpand نشانه بعدی با معنای موقت \relax است. بنابراین زمانی که پردازشگر گسترش نشانه بعدی، \b را می‌بیند، آن را گسترش نمی‌دهد و به لیست نشانه در حال ساخت (متن جایگزین ماکرو) می‌سپارد.

۲. اگر ⟨متغیر نشانه⟩\the داخل تعریف ماکرویی باشد که با \edef تعریف شده باشد، نشانه‌هایی که از ⟨متغیر نشانه⟩\the ساخته می‌شوند، بیشتر گسترش پیدا نمی‌کنند.

۳.۲.۱ نحوه برخورد پردازشگر گسترش با آکولادها (دوابروها)

همانطور که گفتیم آکولادها (دوابروها) نشانه‌های کاراکتری غیر قابل گسترش هستند. در مثال زیر

```
\romannumeral1\number\count2 3{4 ...
```

پردازشگر گسترش همه نشانه‌ها را تا زمانی که } را می‌بیند، گسترش می‌دهد. بنابراین اگر مقدار \count2 ۸ باشد، تک عدد ۱۸۳ را به شکل رومی نشان می‌دهد.

مثال زیر

```
\iftrue {\else }\fi
```

همانند

```
\iftrue a\else b\fi
```

است که یک نشانه کاراکتری مستقل از رده‌اش را پس می‌دهد.
آکولادها (دوابروها) در گسترش ماکرو مهم هستند:

۱. اگر یک گروه از نشانه‌ها داخل آکولاد (دوابرو) قرار بگیرد، به عنوان یک آرگومان در نظر گرفته می‌شود. اگر یک ماکرو یک آرگومان داشته باشد:

```
\def\macro#1{ ... }
```

آرگومان می‌تواند تنها یک نشانه باشد:

```
\macro 1          \macro\$
```

هر چند اگر آرگومان بیشتر از یک نشانه (یک گروه از نشانه‌ها) باشد، باید از گروه‌بندی استفاده کرد تا پردازشگر گسترش نشانه‌ها را به عنوان یک نشانه در نظر بگیرد:

```
\macro{abc}          \macro{d{ef}g}
```

۲. زمانی که پردازشگر گسترش، آرگومان‌های یک ماکروی با پارامتر را می‌خواند، عبارات با آکولادهای (دوابروهای) نامتعادل را در اینجا قبول نمی‌کند. در مثال زیر

```
\def\a#1\stop{ ... }
```

آرگومان، همه نشانه‌ها تا اولین \stop (که داخل آکولاد (دوابرو) نیست) است. بنابراین اگر ماکروی \a به شکل زیر استفاده شده باشد:

```
\a bc{d\stop}e\stop
```

آرگومان \a عبارت bc{d\stop}e است. در حقیقت پردازشگر گسترش تا زمانی که اولین \stop که داخل آکولاد (دوابرو) نیست را ببیند، تمام نشانه‌ها را به عنوان آرگومان \a در نظر می‌گیرد. پردازشگر گسترش در اینجا تنها عبارات متعادل را، قبول می‌کند.

۳.۱ پردازشگر اجرایی

پردازشگر اجرایی لیست‌های افقی، عمودی، و ریاضی را می‌سازد. به همین دلیل، پردازشگر اجرایی در حالت‌های افقی، عمودی، و ریاضی کار می‌کند. هر کدام از این سه حالت، خودشان حالت داخلی و خارجی دارند. علاوه بر ساختن این لیست‌ها، پردازشگر اجرایی پردازشهایی که مستقل از حالت است (مانند واگذاری‌ها)، را هم انجام می‌دهد.

خروجی پردازشگر گسترش، یک رشته از نشانه‌های غیر قابل گسترش است، که توسط پردازشگر اجرایی، پردازش می‌شود. از دید پردازشگر اجرایی، این نشانه‌ها دو دسته هستند:

- نشانه‌هایی که علامت واگذاری (از جمله تعریف ماکروها) هستند، و نشانه‌هایی که علامت کارهایی هستند که مستقل از حالت هستند، مانند \show و \aftergroup.
- نشانه‌هایی که لیست‌ها را می‌سازند: کاراکترها، کادرها، و چسب. روشی که این نشانه‌ها بکار می‌روند، بستگی به حالت فعلی دارد.

بعضی چیزها در هر حالتی می‌توانند به کار بروند. به عنوان نمونه، جعبه‌ها می‌توانند در لیست‌های افقی، عمودی، و ریاضی به کار بروند. اثر این چیزها، البته به حالتی که در آن بکار می‌روند، بستگی دارد. چیزهای دیگر تنها در یک حالت خاص می‌توانند بکار بروند. به عنوان نمونه، نشانه‌های کاراکتری که به رده ۱۱ و ۱۲ تعلق دارند، رابطه نزدیکی با حالت افقی دارند. زمانی که پردازشگر اجرایی این کاراکترها را در حالت عمودی می‌بیند، به حالت افقی سویچ می‌کند.

همه کاراکترها علامت کاراکترهایی که حروف‌چینی می‌شوند نیستند. پردازشگر اجرایی همچنین می‌تواند کاراکترهای شروع و پایان محیط ریاضی (بصورت پیش‌فرض $) و کاراکترهای شروع و پایان گروه ({ و } بصورت پیش‌فرض) را ببیند. کاراکترهای شروع و پایان محیط ریاضی به تک اجازه می‌دهند تا داخل یا خارج محیط ریاضی شود، و آکولادها (دوابروها) به تک اجازه می‌دهند که داخل یا خارج یک سطح جدید گروه‌بندی شود.

دستور \relax غیر قابل گسترش است اما زمانی که پردازشگر اجرایی آن را می‌بینید، کاری انجام نمی‌دهد. دو مثال زیر را مقایسه کنید:

```
\count0=1\relax 2
```

```
\def\empty{}
\count0=1\empty 2
```

در مثال اول، روند گسترش با مشاهده \relax متوقف می‌شود، بنابراین مقدار \count0 ۱ خواهد بود اما در مثال دوم \empty به چیزی گسترش پیدا نمی‌کند، بنابراین مقدار \count0 ۱۲ خواهد بود.

۴.۱ پردازشگر دیداری

پردازشگر خروجی تک الگوریتم‌هایی دارد که کاربر کنترل مستقیمی روی آن‌ها ندارد. این الگوریتم‌ها شکستن پاراگراف، تراز کردن، شکستن صفحه، حروف‌چینی ریاضی، و ساختن فایل pdf. هستند. پارامترهایی مختلفی عملکرد این الگوریتم‌ها رو کنترل می‌کنند.

بعضی از این الگوریتم‌ها نتیجه کارشون رو به شکلی آماده می‌کنند که توسط پردازشگر اجرایی می‌تواند به کار برود. به عنوان نمونه، یک پاراگرافی که به چندین سطر شکسته شده، بصورت یک دنباله از کادرهای افقی با چسب و جریمه‌های متوسط به لیست عمودی اصلی، اضافه می‌شود. همچنین الگوریتم شکستن صفحه، نتیجه کارش را در کادر \box255 ذخیره می‌کند، بنابراین روال خروجی می‌تواند نتیجه کار الگوریتم شکستن صفحه را آنالیز کند. از طرفی، یک فرمول ریاضی نمی‌تواند تکه تکه شود، و طبیعتاً، فرستادن یک کادر به فایل pdf. امری غیر قابل برگشت است.

۵.۱ مثال‌ها

۱.۵.۱ قلم‌انداختن فاصله‌ها

فاصله‌های قلم‌انداخته شده تصویر خوبی از اینکه مراحل مختلف پردازش تک، ورودی تکمیل شده مرحله قبل را قبول می‌کنند، به ما نشان می‌دهند. مثال زیر

```
\def\a{\penalty200}
\a 0
```

معادل 0 \penalty200 نیست که ما انتظار داریم تک یک جریمه به اندازه ۲۰۰ وارد کند و سپس عدد ۰ را حروف‌چینی کند. نتیجه این مثال \penalty2000 است. دلیلش این است که پردازشگر ورودی فاصله بعد از \a را قلم می‌اندازد و در نتیجه مراحل بعدی پردازش \a0 را دریافت می‌کنند نه 0 \a را.

۲.۵.۱ کمیت‌های داخلی و نمایششان

تک از انواع کمیت‌های داخلی مانند اعداد صحیح و بعدها، استفاده می‌کند. این کمیت‌های داخلی یک نمایش خارجی هم دارند که یک رشته از کاراکترها هستند مانند 4711 یا 91.44cm.

تبدیل بین مقدار داخلی و نمایش خارجی، بسته به این که این تبدیل از مقدار داخلی به نمایش خارجی یا از نمایش خارجی به مقدار داخلی صورت می‌گیرد، در مراحل مختلف پردازش صورت می‌گیرد. یک رشته از کاراکترها به یک مقدار داخلی در واگذاری‌هایی مانند

```
\pageno=12          \baselineskip=13pt
```

یا عباراتی مانند

```
\vskip 5.71pt
```

تبدیل می‌شوند و همه این عبارات توسط پردازشگر اجرایی، پردازش می‌شوند.

از طرفی، تبدیل مقادیر داخلی به یک نمایش خارجی بصورت یک رشته از کاراکترها توسط پردازشگر گسترش انجام می‌شود. به عنوان نمونه همه

```
\number\pageno          \romannumeral\year          \the\baselineskip
```

توسط پردازشگر گسترش، پردازش می‌شوند.
به عنوان مثال آخر، اگر \count2=45 باشد، عبارت زیر را در نظر بگیرید:

```
\count0=1\number\count2 3
```

پردازشگر گسترش \number\count2 را پردازش می‌کند و کاراکتر ۴۵ را پس می‌دهد. فاصله‌ای که بعد از ۲ قرار دارد، واگذاری عدد را پایان نمی‌دهد، این فاصله تنها حائل \count2 است تا \count2 با \count23 اشتباه نشود. در مرحله بعدی پردازش، پردازشگر اجرایی \count0=1453 را می‌بیند و آن را اجرا می‌کند.

۲ کدهای رده و حالات داخلی