TEXniki programowania

☞

O pewnych konstrukcjach warunkowych i iteracyjnych

Marcin Woliński

Artykuł składa się z dwóch części. Pierwsza stanowi przegląd metod radzenia sobie z pewną sytuacją, na którą można się natknąć używając TEXowych \ifów. Jest to bardzo wyczerpujący (psychicznie) przegląd. Porażonemu nudą części pierwszej Czytelnikowi nasunie się zapewne pytanie, czy to wszystko czemuś służy. (Niezadowalającą) odpowiedź na to pytanie daje część druga wskazująca na związek z implementacją w TEXu pętli. Mam również nadzieję przedstawić kiedyś Czytelnikom Biuletynu bardziej subtelne konsekwencje tych rozważań.

Zabawa w chowanego

Wyobraźmy sobie, że chcemy zdefiniować makro przypominające LATEXowe \section. Interesuje nas tutaj wyłącznie potrzeba dostosowywania zachowania makra do tego, czy po nim następuje gwiazdka (wtedy jest to nienumerowana sekcja), czy też nie. To co przedstawimy, jest tylko luźno związane ze sposobem, w jaki LATEX definiuje to makro.

Przyjmijmy, że w jakiś magiczny sposób udało się zdefiniować \section tak, że „podgląda" ono następny znak z wejścia nie usuwając go i nie pobierając żadnych parametrów, i że makro \nc zawiera ten znak. Załóżmy, że zdefiniowaliśmy makra \starsection i \normalsection, które wykonają pracę w przypadku „gwiazdkowym" i „niegwiazdkowym". Pozostaje zatem proste podjęcie decyzji na podstawie wartości \nc:

```
\if*\nc \starsection
\else \normalsection \fi
```

Tu jednak pojawia się trudność: chcemy, aby makra \starsection i \normalsection miały parametr — jest nim tytuł rozpoczynanego punktu. Przy zastosowaniu powyższego kodu parametrem \starsection stanie się \else, zaś parametrem \normalsection stanie się \fi. Jest to właśnie zjawisko, któremu chcemy poświęcić chwilę uwagi. W dalszym ciągu pokażemy rozmaite sposoby radzenia sobie w takiej sytuacji.

1. Szkoła klasyczna (t.j. *The TEXbook*) zaleca następujące rozwiązanie:

```
\if*\nc \let\next\starsection
\else \let\next\normalsection \fi
\next
```

W obrębie konstrukcji warunkowej nadaje się wyłącznie wartość makru \next, które później służy do „wyniesienia" jej poza warunek. Zatem parametrem makra \starsection lub \normalsection stanie się to, co stoi dalej (po końcowym \next, czyli po \section). Ewentualna gwiazdka zgodnie z naszym wcześniejszym założeniem została na wejściu i musi być „połknięta" przez \starsection, jednak \starsection jest wywoływane tylko jeżeli gwiazdka rzeczywiście wystąpiła, więc może ją połykać w ramach poszukiwania obowiązkowych argumentów.

2. Ponieważ są sytuacje, w których TEX nie wykonuje przypisań, powyższe rozwiązanie nie zawsze daje się zastosować. Ponadto przypisania często powodują spowolnienie wykonywania programu. Stąd bierze się szkoła mniej klasyczna (t.j. Szkoła Unikania Przypisań), która proponuje coś takiego:

```
\if*\nc \expandafter\starsection
\else \expandafter\normalsection \fi
```

Polecenie \expandafter powoduje zmianę kolejności rozwijania makr: najpierw zostanie rozwinięte drugie po \expandafterze. Jeśli warunek był prawdziwy i wykonywana jest pierwsza gałąź konstrukcji, \expandafter „trafi" w \else. Rozwinięcie \else polega na pominięciu wszystkich leksemów[1] aż do pasującego \fi, zatem gdy TEX zacznie poszukiwać parametru dla \startsection, po \else ...\fi nie będzie już śladu. W przypadku wykonania drugiej gałęzi \expandafter rozwija \fi, które jest po prostu pomijane, a TEX odnotowuje sobie, że już nie jest wewnątrz warunku.

3. W naszym przykładzie makra \starsection i \normalsection musiałyby być bardzo podobne. Zapewne wygodniej byłoby mieć jedno makro, które jako parametr dostaje informację, czy sekcja jest numerowana, czy też nie. Stosowanie rozwiązania podobnego do powyższego w tym przypadku wymaga użycia większej liczby \expandafterów: po jednym przed każdym leksemem każdej z gałęzi warunku.

1: Idąc śladem specjalistów od kompilatorów używam słowa leksem jako tłumaczenia angielskiego *token*.

```
\if*\nc \expandafter\dosection
        \expandafter\notnumbered
\else \expandafter\dosection
      \expandafter\numbered
\fi
```

Każdy z \expandafterów w takim łańcuszku „odpala” następny, ostatni zaś rozwija \else bądź \fi. Dopiero wtedy TEX bierze się za oglądanie leksemów, które były po drodze.

4. Jeżeli zechcemy, aby \dosection otrzymywało zamiast sztucznych znaczników przepis na wyprodukowanie numeru sekcji (czasem pusty), ilość \expandafter zrobi się astronomiczna. Możemy sobie także wyobrazić, że w jakimś zastosowaniu w ogóle nie wiemy, ile leksemów znajdzie się w gałęzi warunku, i w związku z tym nie umiemy odpowiednio powstawiać \expandafterów. Wówczas możemy powrócić do rozwiązania klasycznego, używając \def zamiast \let:

```
\if*\nc
  \def\next{\dosection
        {\usecounter{section}}}%
\else \def\next{\dosection{}}\fi
\next
```

5. Możemy jednak nie poddawać się tak łatwo. Załóżmy, że mamy pomocnicze makra

```
\def\firstoftwo#1#2{#1}
\def\secondoftwo#1#2{#2}
```

Możemy teraz postąpić tak

```
\if*\nc \expandafter\firstoftwo
\else \expandafter\secondoftwo \fi
{\dosection{\usecounter{section}}}
{\dosection{}}
```

Poprzednia sztuczka powoduje, że \firstoftwo i \secondoftwo widzą właściwe parametry, makra zaś wybierają odpowiednią z następujących dalej możliwości.

6. Te dwie czynności możemy skombinować w jedną przy pomocy następującego makra

```
\def\afterfi#1#2\fi{\fi #1}
```

Makro \afterfi połyka niepotrzebną część warunku, ograniczoną przez \fi, wstawia z powrotem \fi, które też zostało połknięte i wreszcie wykonuje akcję, która jest jego pierwszym parametrem. Zwróćmy uwagę, że \afterfi nie dałoby się użyć w zagnieżdżonym warunku, ponieważ nie umie ono rozpoznać, czy \fi ograniczające argument należy do właściwego \if.

```
\if*\nc
    \afterfi{\dosection
              {\usecounter{section}}}%
\else \afterfi{\dosection{}}\fi
```

Jeśli warunek jest prawdziwy, pierwsze \afterfi zjada fragment warunku od \else aż do \fi (a wstawia \fi, ale dla TEXa jest to nieistotne — ważne, że warunek się skończył). Jeżeli wartością jest fałsz, TEX skacze do \else w ogóle nie zauważając pierwszego \afterfi, a do akcji wkracza drugie. Ma ono do połknięcia tylko \fi, zatem efektywnie przestawia ono miejscami \fi i swój pierwszy argument. Pierwszy argument \afterfi jest ograniczony nawiasami klamrowymi, zgodnie ze zwyczajami TEXa znikają one w procesie dopasowywania parametrów.

Mniej więcej takich nieprzyjemności możemy się spodziewać po makrze \section. Jednak życie jest niepomiernie bardziej bogate oferując nam wielokrotnie zagnieżdżone warunki, w których czasami też chcemy się bawić w chowanego.

Żeby nie wymyślać nowego przykładu, wyobraźmy sobie, że \section ma trzecią wersję, jeżeli po nim nastąpi plus (\section+). Cóż możemy począć w takiej sytuacji?

7. Oczywiście rozwiązanie klasyczne zawsze jest chętne do współpracy

```
\if*\nc \let\next\starsection
\else\if+\nc \let\next\plussection
  \else \let\next\normalsection
  \fi\fi
\next
```

8. Rozwiązanie z \expandafter wymaga znacznie większej liczby zamian

```
\if*\nc \expandafter\starsection
\else\if+\nc
      \expandafter\expandafter
      \expandafter\plussection
  \else \expandafter\expandafter
        \expandafter\normalsection
  \fi\fi
```

Prześledźmy dla przykładu gałąź warunku dla \plussection. Pierwszy \expandafter odkłada „na bok” drugi i odpala trzeci. Trzeci odkłada \plussection i rozwija \else. To powoduje połknięcie gałęzi dla \normalsection i zamykającego ją pierwszego \fi. Teraz TEX wstawia z powrotem leksemy odłożone „na bok” i widzi \expandafter\plussection\fi. Przywrócony z niebytu

`\expandafter` rozwija drugie `\fi` i wreszcie do akcji może wkroczyć `\plussection`.

9. Jeżeli akcje do wykonania w gałęziach warunku mają większą ilość leksemów, możemy zastosować sztuczkę z makrami `\firstofthree`, `\secondofthree` i `\thirdofthree` podobnymi do `\firstoftwo` i `\secondoftwo`. Zamiast jednak stosować `\expandafter` jak powyżej, możemy do kolejnej wariacji wprowadzić nowego solistę:

```
{\if*\nc \aftergroup\firstofthree
 \else\if+\nc
         \aftergroup\secondofthree
   \else \aftergroup\thirdofthree
   \fi\fi}
{\dosection{\usecounter{section}}}%
{\dosection{\strange{option}}}%
{\dosection{}}
```

Wokół warunków została dodana nadmiarowa grupa. Kolejny Dziwny Operator — `\aftergroup` służy tutaj do wyrzucenia makra wyłuskującego właściwą akcję poza tę grupę i tym samym poza warunki.

Ponieważ `\aftergroup` musi zostać wykonany a nie rozwinięty, to rozwiązanie nie nadaje się do sytuacji „tylko rozwinięciowych". Czy zatem ma ono jakiekolwiek znaczenie praktyczne? Koniec dodatkowej grupy unieważnia przypisania wykonane w jej wnętrzu, co pozwala zdefiniować pomocnicze wartości dla porównania nie zakłócając zewnętrznego środowiska. W ten sposób można napisać np. wersję `\@ifnextchar`, która nie zostawia po sobie śmieci w postaci pomocniczych makr.

Obszerna dygresja. Dają się pomyśleć zastosowania w których występuje częste sprawdzanie czy pewne makra są zdefiniowane. Co więcej może się zdarzyć, że makra takie mają nazwy tworzone w locie przy pomocy operatora `\csname`. Operator ten ma tę zabawną cechę, że jeżeli makro nie było wcześniej zdefiniowane zostanie mu przez `\csname` przypisana wartość `\relax`. Jest to bardzo szczególne przypisanie. Dokonuje się ono nawet w momentach gdy TeX „tylko rozwija". Np. jeżeli `\foo` nie było wcześniej znane, w konstrukcji

```
\edef\bar{\csname foo\endcsname}
\show\foo
```

`\show` oświadczy, że `\foo` ma wartość `\relax`. Ponadto przypisanie to jest lokalne, nawet jeżeli `\globaldefs=1`. To powoduje, że

```
\global
\expandafter\def\csname foo\endcsname{%
  jakieś tam makro}
```

powoduje odłożenie lokalnej wartości `\foo` na *save stack*, ponieważ następują *dwa* przypisania: najpierw lokalne, potem globalne. Jedynym sposobem na zabezpieczenie przed przepełnieniem tego stosu przez długą sekwencję tego rodzaju definicji jest zastąpienie ich przez

```
{\global
\expandafter\def\csname foo\endcsname{%
  jakieś tam makro}
}
```

Podobne kombinacje dzieją się gdy wykonujemy nielokalne przypisania w zależności od wyniku testów wykonywanych na makrze skonstruowanym przy pomocy `\csname`. W takim przypadku konstrukcja numer 9 pozwala natychmiast po ustaleniu wartości warunku odśmiecić *save stack*. (Koniec obszernej dygresji)

10. Można sobie również poradzić stosując makra analogiczne do `\afterfi` z argumentem ograniczonym dwoma `\fi`.

```
\def\afteriffifi#1#2\fi\fi{\fi #1}
\def\afterfifi#1#2\fi\fi{\fi\fi #1}
```

Pierwsze z nich powinno być wołane z zewnętrznego `\ifa`. Jego argument jest ograniczony dwoma `\fi`, ale wstawia ono tylko jedno, bo tylko jeden warunek został rozpoczęty do chwili jego wywołania. Drugie makro służy do wynoszenia akcji z wewnętrznego warunku, zatem dostawia `\fi\fi` kończące oba warunki.

```
\if*\nc
  \afteriffifi
    {\dosection{\usecounter{section}}}
\else\if+\nc
  \afterfifi
    {\dosection{\strange{option}}}
  \else
  \afterfifi{\dosection{}}
  \fi\fi
```

11. Liczbę `\expandafter`ów, które są potrzebne w pierwszym rozwiązaniu nieklasycznym, możemy zmniejszyć poprzez przeniesienie wszystkich akcji na pierwszy poziom zagnieżdżenia warunków:

```
\ifcase
  \if*\nc0 %
  \else\if+\nc1 %
```

```
      \else2 \fi\fi
    \expandafter\starsection \or
    \expandafter\plussection \or
    \expandafter\normalsection \fi
```

Dodajemy tutaj zewnętrzny `\ifcase`, a zagnieżdżone warunki produkują numer wybranego przypadku zamiast akcji do wykonania. Zwróćmy uwagę, że po każdej cyfrze znajduje się spacja powodująca, że TEX kończy poszukiwanie liczby (spacja zostaje połknięta jako część liczby). Gdyby zabrakło którejś z nich, TEX rozwinąłby tekst pierwszego przypadku ciągle nie znając wartości dla `\ifcase` i byłaby katastrofa. Pojedynczy `\expandafer` wystarcza do rozwinięcia `\or` lub `\fi` kończącego warunek przed rozwijaniem dalszych makr (rozwijanie `\or` wygląda tak jak rozwijanie `\else`, ale może pochłonąć wiele przypadków za jednym zamachem).

12. Ten przykład można skrzyżować z techniką `\afterfi`, przez co uzyskujemy moją ulubioną wersję:

```
\ifcase
  \if*\nc0 %
    \else\if+\nc1 %
       \else2 \fi\fi
  \afterfi{\dosection
      {\usecounter{section}}}
\or
  \afterfi{\dosection
      {\strange{option}}}
\or
  \afterfi{\dosection{}}
\fi
```

Zaletą tego rozwiązania jest to, że mając tylko jedno pomocnicze makro `\afterfi` możemy poradzić sobie z dowolnie zagnieżdżonymi warunkami i dowolnie złożonymi akcjami. Niektóre z wewnętrznych `\if`ów mogą oczywiście produkować te same numery przypadków, a stosując techniki proponowane przez J. Fine'a w [3], możemy skonstruować dowolnie złożone funkcje logiczne. Wadą jest skomplikowany zapis (trzeba się nie pomylić numerując przypadki).

13. Na zakończenie, żeby zupełnie zamącić w głowach tym, którzy dotrwali aż do tego miejsca, fragment kodu do przeanalizowania. Zakładamy, że warunek `\ifA` został zdefiniowany przy pomocy `\newif`. Co zostanie wydrukowane w zależności od wartości tego warunku?

```
{\ifA La\aftergroup\else \TeX}nik\fi
```

Multum `\loop`orum

Przyjrzyjmy się teraz TEXowej pętli. Fakt, że konstrukcja iteracyjna nie jest wbudowana w TEXa, był uroczym zaskoczeniem dla wielu początkujących (na przykład dla mnie). Choć właściwie nie ma tu żadnej magii — możliwość definiowania rekurencyjnych makr wystarcza do „zrobienia” iteracji.

Ciekawsza jest metoda pozwalająca w definicji pętli zastosować tzw. rekursję końcową (*tail recusion*). W sposobach dokonania tej sztuki możemy dostrzec echa naszych wcześniejszych rozważań.

Przypomnijmy składnię konstrukcji iteracyjnej zdefiniowanej przez plain TEXa:

```
\loop
 <akcje>
 <warunek>
 <akcje>
\repeat
```

Warunek jest dowolnym TEXowym `\if`em bez zamykającego go `\fi`.

Definicja pętli zaczyna się od „lukru syntaktycznego” realizującego tę składnię

```
\let\repeat\fi
\def\loop#1\repeat{%
  \def\body{#1}\iterate}
```

Ponieważ w konstrukcji `\loop ... \if ... \repeat` znajduje się niezbilansowany `\if`, pętla taka powodowałaby katastrofę, gdyby znalazła się w pomijanym fragmencie jakiegoś warunku. Tajemnicze pierwsze przypisanie powoduje, że `\repeat` wygląda jak `\fi` i jeżeli TEX przeskakuje przez całą konstrukcję, bilansuje `\if` w treści pętli. W przeciwnym przypadku znaczenie związane z leksemem `\repeat` nie jest istotne (równie dobrze mógłby być niezdefiniowany), ponieważ służy on wyłącznie jako ogranicznik treści pętli i znika w procesie poszukiwania argumentu dla `\loop`.

W definicji `\loop` zapamiętujemy treść pętli (z niepełnym `\if`em) w makrze `\body` i wywołujemy makro `\iterate`.

Spróbujmy teraz naiwnej definicji `\iterate`

```
\def\iterate{\body\iterate\fi}
```

Konstrukcja ta będzie poprawnie działać, jeżeli pętla nie wykona się zbyt wiele razy. Ponieważ `\body` rozwija się do czegoś w rodzaju `...\if...`, więc wywołanie `\iterate` spowoduje wykonanie części treści przed warunkiem, a następnie obliczenie warunku. Jeżeli warunek jest fałszywy, cały dalszy ciąg zostanie pominięty i pętla się zakończy, jeżeli zaś prawdziwy wykona się reszta treści pętli i ponownie (rekurencyjnie) zostanie wywołane `\iterate`.

Rekurencyjne wywołanie `\iterate` zostaje wykonane *przed* rozwinięciem końcowego `\fi`, zatem w miarę kolejnych wywołań za `\iterate` będzie się gromadził coraz dłuższy „ogon” leksemów `\fi`. Ten ogon może spowodować przepełnienie pamięci TeXa, jeżeli pętla będzie się „obracać” odpowiednio długo. Zwróćmy uwagę, że owe `\fi` nie są do niczego potrzebne, czekają po prostu na pominięcie. Warto więc skłonić TeXa do rozwinięcia (czyli pominięcia) `\fi`, *zanim* wywoła po raz kolejny `\iterate`. Jest to to samo zjawisko, z którym mieliśmy do czynienia poprzednio.

W literaturze TeXowej możemy znaleźć trzy definicje pętli.
Definicja profesora Knutha ([1] s. 352) jak można się spodziewać reprezentuje Szkołę Klasyczną:

```
\def\iterate{\body\let\next=\iterate
  \else\let\next=\relax\fi\next}
```

Historycznie druga definicja należąca do Kabelschachta [2] reprezentuje Szkołę Unikania Przypisań:

```
\def\iterate{%
  \body\expandafter\iterate\fi}
```

Jest wreszcie definicja Keesa van der Laana [4]. Tutaj rekurencyjne wywołanie `\iterate` następuje naprawdę *po* zakończeniu warunku.

```
\def\iterate{%
  \body\else\etareti\fi\iterate}
\def\etareti\fi\iterate{\fi}
```

Definicja ta nawiązuje do stosowanego przez van der Laana FIFO. Drugi `\def` (wyglądający nieco dziwnie) tworzy makro o nazwie `\etareti`, charakteryzujące się tym, że przy każdym jego użyciu dalej muszą następować leksemy `\fi` oraz `\iterate`, które zostaną pominięte. Jest to zatem specjalizowane makro z gatunku `\afterfi`.

Mała dygresja. W FIFO zawsze mnie interesowało, dlaczego Laan nie doprowadził do końca dzieła separacji operacji przechodzenia po liście od przetwarzania elementów. Mianowicie akcja do wykonania na każdym elemencie listy jest zadawana poprzez *przypisanie* wartości makru `\process` zaszytemu w `\fifo`, podczas gdy mogłaby ona być parametrem. Wówczas FIFO symulowałoby znaną z języków z funkcjami wyższego rzędu operację *map*.

Pętla Kabelschachta wyróżnia się spośród pozostałych tym, że nie jest dokładnie równoważna pętli plainowej. Pozwala ona na konstrukcje typu:

```
\loop ... \if.. ...\else ... \repeat
```

W tej postaci oczywiście nie jest atrakcyjna możliwość wpisania czegoś między `\if` a `\else` (ten fragment wykona się tylko raz przy wyjściu z pętli). Atrakcyjne jest efektywne negowanie warunku pętli, np.

```
\loop
  ...
\ifeof\inputstream\else\repeat
```

może wykonywać przetwarzanie pliku aż do napotkania na jego koniec. Taka operacja nie daje się elegancko zapisać przy pomocy pętli z plain’a (zobacz rozpaczliwe usiłowania van der Laana GUST nr 1 s. 32).

Warto wiedzieć, że LaTeX2ε używa tej właśnie definicji.

Bibliografia

1. Donald E. Knuth *The TeXbook*.
2. A. Kabelschacht `\expandafter` *vs.* `\let` *and* `\def` *in conditionals and a generalization of plain's* `\loop`. *TUGboat*8#2, 184–185.
3. Jonathan Fine *Fun with* `\if` Biuletyn GUST nr 3, s. 45–46.
4. Kees van der Laan *FIFO and LIFO sing the BLUes* Biuletyn GUST nr 4, s. 20–26.

◇ Marcin Woliński
Wolinski@bull.mimuw.edu.pl

Od red.: Operację *map*, o której wspomina Autor, zaimplementował w swoich makrach Marek Ryćko. Wykorzystywana jest ona w przedstawionych w Bachotku makrach licz, tsp i tun, dostępnych na serwerze GUST.

(*StaW*)