GUST

Grupa Użytkowników Systemu TEX

Zeszyt 1 (1993)

Biuletyn Polskiej Grupy Użytkowników Systemu TEX
Zeszyt 1 (1993)

Gdańsk
Marzec 1993

Redakcja:
Włodek Bzyl
Tomasz Przechlewski

Adres:
Instytut Matematyki UG
Wita Stwosza 57, 80-952 Gdańsk
matwb@halina.univ.gda.pl

Zamiast wstępu

Hanna Kołodziejska
prezes GUST-u

Chciałabym podziękować wszystkim, dzięki którym powstał GUST. Występując z nieśmiałym pomysłem powołania polskiej grupy użytkowników TEX-a na jesieni 91 roku nie spodziewałam się aż tak przychylnego i powszechnego odzewu. Prawie 40 osób przyjechało 8 stycznia na spotkanie założycielskie, ponad 60 na I Walne Zebranie w dniu 5 czerwca. Dla większości była to całodniowa wyprawa do Warszawy.

Chciałabym także wymienić kilka osób, których nazwiska szczególnie kojarzę z historią TEX-a w Polsce. Pierwszym znanym mi miejscem, do którego dotarł TEX był Instytut Informatyki Uniwersytetu Warszawskiego. W 1983 roku prof. Jan Madey, obecnie dyrektor Instytutu, z własnych środków zakupił w USA taśmę dystrybucyjną z TEX-em 0.8. Piotr Carlson wykonał decydującą część prac nad uruchomieniem otrzymanego TEX-a na komputerze IBM 370 pod systemem VM/CMS. Janusz S. Bień cały czas zabiegał o upowszechnienie TEX-a zarówno w Instytucie, jak i poza nim: należąc przez lata do TEX Users Group (przez dłuższy czas jako jedyny z Polski!) umożliwiał nam, innym TEX-owcom, dostęp do bieżącej literatury TEX-owej. Kompletował także i udostępniał docierające do Polski TEX-owe oprogramowanie *public domain*.

Ogromną pracę wykonali Marek Ryćko i Bogusław Jackowski przygotowując i doskonaląc polską adaptację TEX-a znaną najpierw jako LEX, obecnie MEX.

To tylko kilka osób z szerokiego grona tych, których powinnam wymienić. Jestem pewna, że na każdej uczelni, w wielu instytutach naukowych i wydawnictwach były i są osoby, które zrobiły dużo dobrego dla upowszechnienia TEX-a. Odszukajmy ich!

Teraz już jest GUST. Mamy nasz biuletyn, tworzymy archiwum oprogramowania, nawiązujemy kontakty między sobą i z grupami zagranicznymi. Możemy więc śmiało powiedzieć: czas bez-GUŚCIA się skończył.

Szczerze zachęcam wszystkich do współpracy!

W zeszycie

Od Redakcji

Redakcja biuletynu chciałaby podziękować wszystkim osobom, które przyczyniły sie do ukazania biuletynu, a przede wszystkim:

- Toniemu Walterowi za projekt znaczka GUST-u
- autorom prac zamieszczonych w numerze
- Rektorowi Uniwersytetu Gdańskiego za częściowe pokrycie kosztów druku

W drugim numerze powinny pojawić się nowe działy:

- Poznajemy LAMEX
  prowadzony przez Wiesława Pawłowskiego
- Oprogramowanie nie tylko MEXowe
  prowadzony przez Stanisława Wawrykiewicza.

Zachęcam do współpracy. Piszcie na adres:

◇ Włodzimierz Bzyl
Instytut Matematyki UG
Wita Stwosza 57, 80-952 Gdańsk
matwb@halina.univ.gda.pl

PLAIN

Opis pakietu MEX

Bogusław Lichoński

Z wielką przyjemnością przedstawiam pierwszy z prawdziwego zdarzenia dwujęzyczny format dla TEX-a o leśnej nazwie MEXw którym obok języka angielskiego pojawia się język polski.

MEX 1.03

Już 18 grudnia 1992 roku autorzy pakietu — panowie Bogusław Jackowski i Marek Ryćko — udostępnili nam wszystkim wersję 1.03 pakietu MEX. Składa się on z fontów w postaci źródłowych plików METAFONT-owych, zawierających polskie znaki diakrytyczne i używane w języku polskim znaki cudzysłowów, formatów w wersji źródłowej, zwanych MEX i LAMEX (odpowiedników formatów `plain` i `lplain`) oraz polskich wzorców dla TEX-owego algorytmu dzielenia wyrazów. Całość jest oprogramowaniem typu *public domain*.

Oto szczegółowa lista plików pakietu MEX:

- zbiór plików formatu MEX; `mex.tex`, `mex1.tex`, `mex2.tex`, `mexconf.tex`, `polhyph.tex`;
- format LAMEX; `lamex.tex`;
- zbiór plików wsadowych i pomocniczych do generowania obu formatów w środowisku TEX-a 3.x z implementacji emTEX-a. `gm-lamex.bat`, `gm-mex.bat`, `gl-lamex.bat`, `gl-mex.bat`, `maz2pl.tcp`, `maz2pl.tpc`, `lat2pl.tcp`, `lat2pl.tpc`.

Nad pakietem pracowali także: pani Hanna Kołodziejska, autorka polskich wzorców dzielenia wyrazów oraz pan Roman Tomaszewski, wybitny polski typograf, prezes polskiego oddziału A-Typ-I (*Association Typographique Internationale*).

zalecana jest notacja bezprefiksowa, bieżącym językiem jest język polski, 256 znaków w foncie, frenchspacing

Z chwilą rozpoczęcia pracy z formatem MEX obowiązują nieco odmienne zasady niż z formatem `plain`. Po pierwsze bieżącym językiem jest język polski. Po drugie obowiązującym sposobem notacji polskich znaków w plikach źródłowych jest notacja bezprefiksowa (bezciachowa) i może to być standard Mazovia albo Latin 2. Po trzecie stosowanym układem polskich znaków diakrytycznych w fontach jest układ PL. Nowością jest wykorzystanie możliwości TEX-a 3.x i wprowadzenie układu 256 znaków w foncie. Po czwarte obowiązującym sposobem spacjowania po znakach przystankowych jest frenchspacing, czyli wielkość odstępów między wyrazami nie zależy od tego, jaki znak poprzedza odstęp.

Użytkownik może zmienić powyższe zasady modyfikując plik konfiguracyjny `mexconf.tex`, a następnie ponownie wygenerować format.

mamy możliwość tworzenia polskich komend — `\Ściąga`, `\wskazówka`

Mój pierwszy kontakt z MEX-em był dla mnie bardzo miły. Denerwująca mnie od początku używania TEX-a notacja ciachowa przy pisaniu polskich liter została zastąpiona przez wygodne umieszczanie w tekście polskich znaków diakrytycznych zapisywanych jako zwykłe znaki o kodach w preferowanej przeze mnie Mazovii. Pozwala to na bardzo wygodne składanie tekstów! Polskie litery widoczne są na ekranie w trakcie przygotowywania tekstu, ponadto możliwe jest definiowanie polskich komend TEX-owych, jak choćby `\Ściąga`, `\wskazówka`.

```
\prefixing    id/x ku s/lo/ncu
\nonprefixing    idź ku słońcu
```

Aby zachować zgodność z LEX-em (wcześniejszą adaptacją TEX-a, również autorstwa panów BJ & MR) MEX posiada komendę `\prefixing`, która umożliwia pisanie tekstów w notacji prefiksowej (notację bezprefiksową przywraca `\nonprefixing`).

Ponieważ znak '/' w notacji prefiksowej wykorzystany jest do oznaczania polskich liter, konieczne było wprowadzenie konwencji notacyjnej pozwalającej uzyskać w składzie symbol „ciach". W MEX-u przyjęto naturalną zasadę uzyskania znaku „ciach" przez '//'. (Można także używać dłuższej komendy `\normalslash`.) Należy zwrócić uwagę, że w czasie pisania do pliku za pomocą komendy `\write` makra są rozwijane. Dotyczy to w także makra / i trzeba na to zwracać uwagę przy pisaniu do plików, które mają być potem czytane przez TEX-a. Użyteczne też mogą być komendy `\emulateplain` i `\emulateLaTeX` służące do emulacji formatów `plain` i LATEX odpowiednio wewnątrz formatów MEX i LAMEX.

```
\language\english
I wanna go home
\language\polish
Ja chcę iść do domu
```

Zaletą MEX-a jest możliwość przełączania między sposobami dzielenia wyrazów. W praktyce dosyć często w dokumentach pojawiają się fragmenty choćby w języku angielskim. „Uruchomienie" reguł dzielenia wyrazów angielskich dokonujemy komendą `\language\english`. Przełączenie na reguły polskie dokonujemy poleceniem `\language\polish`.

```
I like “Times”
Ja lubię „Wprost”
Je préfère «Le Monde»
```

Autorzy MEX-a nie zapomnieli o polskich cudzysłowach. Bardzo częstym błędem jest używanie w polskich tekstach anglosaskich cudzysłowów. Polski cudzysłów otwierający uzyskać można przez napisanie dwóch znaków `,,`, w składzie uzyskamy „ podobnie jest z cudzysłowem zamykającym, to znaczy pisząc `''` (lub `"`) otrzymamy ". Niekiedy w języku polskim używany jest też inny rodzaj cudzysłowów, tak zwane cudzysłowy «francuskie». Lewy francuski cudzysłów oznaczany jest dwoma znakami mniejszości, a prawy dwoma znakami większości.

Przy kompilacji znaków `,,`, `''`, `<<` i `>>` na odpowiadające im w składzie cudzysłowy wykorzystywany jest TEX-owy mechanizm ligatur. Oczywiście informacje o ligaturach zostały zawarte w METAFONT-owych źródłach fontów PL.

```
\plqq Pani\prqq      „Pani”
\flqq Elle\frqq      «Elle»
```

Mechanizmu wstawiania cudzysłowów za pomocą ligatur nie zastosowano w fontach stosowanych do składu formuł matematycznych oraz fontach nieproporcjonalnych (np. włączane komendą `\tt`). Cudzysłowy można wtedy uzyskać przez użycie komend:

- `\plqq`, `\prqq` — cudzysłowy polskie ,
- `\flqq`, `\frqq` — cudzysłowy francuskie.

```
\layoutpl
\layoutlatintwo
\layoutpone
\layoutmazovia
```

Być może niektórym użytkownikom TEX-a zdarzyło się wykorzystywać w jednym dokumencie fonty różnego pochodzenia, ale czy udawało się robić taki koktajl z fontami o różnych układach polskich znaków? MEX posiada taką możliwość. Możliwe jest to dzięki komendzie `\layout{xxx}`, gdzie `xxx` $\in$ {`pl`, `mazovia`, `latintwo`, `pone`}.

Przedstawiony wyżej zbiór posiada cztery elementy, z których jedynie pone (skrót od polishone czyli P1) może być niezrozumiały. Użycie komendy `\layout{pone}` powoduje przygotowanie MEX-a do pracy z fontami P1, poprzednikami fontów PL. Jak widać MEX przygotowany jest do pracy z fontami w układzie PL jako podstawowymi oraz z fontami zawierającymi polskie litery zgodne z kodem Mazovia, Latin 2 oraz P1. Należy zwrócić uwagę, że komenda `\layout` nie „włącza" samego fontu, jedynie przygotowuje system do pracy z fontami w odpowiednim układzie.

```
W MEX-u polskie znaki mają kategorię „litera”
```

Przyjrzyjmy się dokładnie komendzie `\layout`. Wykonanie jej polega na przypisaniu polskim znakom diakrytycznym kategorii „litera". Jeżeli używamy układu innego niż PL, polskim znakom przypisana zostaje kategoria „aktywny" i traktowane są one jako makra rozwijające się do odpowiednich liter. Oprócz tego dokonuje się odpowiednia zamiana tablicy `\sfcode` oraz utworzenie tablic `\uccode` i `\lccode` zgodnych z układem fontu. W notacji prefiksowej odpowiednio zmieniają się też znaczenia komend `\plqq`, `\prqq`, `\flqq`, `\frqq`. Pozycjom polskich znaków zostają przypisane odpowiednie kody matematyczne. Ważnym efektem jest włączenie reguł przenoszenia wyrazów w języku polskim (zgodnych z układem fontu), jeśli nie ma odpowiednich reguł, włączany jest zakaz dzielenia wyrazów. Zmienia się także znaczenie symbolu `\polish`.

Niezależnie od obowiązującego układu fontu i znaczenia symbolu `\polish`, symbole:

- `\pllanguage`,
- `\mazovialanguage`,
- `\latintwolanguage`,
- `\ponelanguage`

oznaczają numery języków w sensie TEX-a, czyli numery odpowiednich reguł dzielenia wyrazów.

Oczywiście TEX „widzi" zasady dzielenia wyrazów zależne zarówno od języka, jak i od układu

liter w bieżącym foncie. Jeżeli więc użytkownik „włącza" font o innym układzie (innych kodach) polskich znaków diakrytycznych, to towarzyszyć temu powinno przełączenie TEX-a na inne reguły przenoszenia wyrazów.

Podczas generowania formatu `mex.fmt` domyślnie wbudowywane są reguły przenoszenia wyrazów w języku angielskim i polskim.

Użytkownik — zależnie od wielkości dostępnej pamięci w komputerze — może przystosować format do korzystania z reguł przenoszenia dla innych układów polskich fontów. Wystarczy dokonać prostej modyfikacji pliku `mexconf.tex`.

```
żółto\=niebieski                    żółto-
                                    -niebieski
```

Polskie reguły ortograficzne są czasami sprzeczne z anglosaskim. Na przykład zaleca się by wyrazy łączone, takie jak „żółto-niebieski", przenosić w miejscu łączenia. W języku angielskim nie powtarza się łącznika na początku drugiego wiersza. Ale w polskim dywiz należy powtórzyć na początku nowej linii. Dlatego w polskiej wersji została wprowadzona komenda `\=`, która użyta w miejsce łącznika (żółto-niebieski) powoduje dzielenie zgodne z opisaną regułą. Ponieważ jednak w formacie `plain` komenda `\=` oznacza akcent „macron" dlatego format MEX zastępuje tę niedogodność makrem `\macron`.

$$x_{\text{średnia arytmetyczna}} \geqslant x_{\text{średnia geometryczna}}$$
$$a\frac{x}{\ln x} \leqslant \pi(x) \leqslant b\frac{x}{\ln x}$$

MEX dba również o matematykę. Co prawda modyfikacje w stosunku do formatu `plain` są tu niewielkie, jednak interesujące. Nowością jest używanie polskich liter (w dowolnej notacji) w formułach matematycznych w indeksach górnych i dolnych. Stosowane w Polsce znaki relacji mniejsze-równe ($\leqslant$) i większe-równe ($\geqslant$) różnią się od znaków zaprojektowanych przez D. E. Knutha. Fonty PL (ściślej PLSY) zawierają polskie znaki relacji. Uzyskujemy je za pomocą komend: `\xleq` lub `\xle` oraz `\xgeq` lub `\xge`.

```
„polskie" funkcje: \tg, \ctg, \tgh, \ctgh
oraz \arc\sin, \arc\cos.
```

Zauważmy, że w Polsce używa się do nazwania funkcji trygonometrycznych symboli „tg", „ctg", „tgh" i „ctgh" a nie „tan", „cot", „tanh" i „coth". Twórcy MEX pomyśleli i o tym tworząc komendy `\tg`, `\ctg`, `\tgh`, oraz `\ctgh`. Jest to niby tylko zmiana nazwy, jakże jednak kojarząca się nam ze swojskim zapisem. Dodatkowo wprowadzona została komenda `\arc`, która w połączeniu z powyższymi (`\arc\sin`, `\arc\cos`, itd.) daje w składzie $\arcsin$, $\arccos$ itd. Jak widać między słowem „arc" i nazwą funkcji istnieje odpowiedni odstęp.

Na zakończenie warto wspomnieć, że obok formatu MEX pakiet zawiera także format LAMEX, który jest polskim odpowiednikiem angielskiego formatu `lplain`. Autorzy MEX-a zadbali więc o wszystkich zwolenników LATEX-a.

Podsumowując, MEX okazał się bardzo wygodnym narzędziem do składania polskich tekstów. Nie jest oczywiście ideą nową, tego typu formaty już dawno powstały w innych krajach. MEX zbliżył nas więc nieco do Europy, dając komfort pracy. Aby nie tworzyć pustych zdań powiem, że MEX bardzo podoba mi się, używam go od lutego 1992 roku w środowisku implementacji emTEX. Takie połączenie wydaje mi się bardzo dobre i mogę je wszystkim polecić.

Formaty MEX i LAMEX wygenerowałem przy użyciu kompilatora `tex286.exe` wersja 3.0 [3a], zaś źródła fontów PL skompilowałem METAFONT-em `mf286.exe` wersja 2.0 [3a], przy użyciu programu `mfjob.exe` wersja 1.1f. Wszystkie powyższe programy pochodzą z pakietu emTEX-a E. Mattes'a.

METAFONT

Zabawy z METAFONT-em

Tomasz Przechlewski

Tekst ten ma stanowić zachętę do zajęcia się METAFONT-em. Oczywiste jest, że trudne będzie generowanie własnych fontów — nie wystarczy tutaj znajomość samego programu. Wystarczy ona jednak z pewnością do projektowanie prostych symboli geometrycznych czy modyfikacji istniejących czcionek a od czasu do czasu tego typu rzeczy są wielu z nas potrzebne. Z własnego doświadczenia wiem, jak trudne może być rozpoczęcie pracy z METAFONT-em stąd starałem się zamieścić w tekście dużo informacji, które — mam nadzieję — pozwolą bezproblemowo zainstalować program.

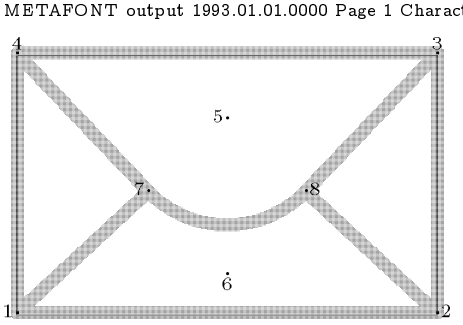
Zaczniemy od czegoś prostego, ale równocześnie choć trochę użytecznego, mianowicie zaprojektujemy znak koperty (patrz rysunek 1). Po pierwsze utwórzmy plik `koperta.mf` i wpiszmy za pomocą naszego ulubionego edytora tekstowego taki oto dziwny kod:

```
1. golden_ratio:=.5(sqrt(5)-1);
2. if unknown size#: size#=36pt#; fi
3. if unknown p#: 20p#= fi
4. golden_ratio*ew#=eh#=size#;
5. if unknown a.top: a.top=.75; fi
6. if unknown a.bot: a.bot=.15; fi
7. mode_setup; p:=round(p#*hppp);
8. beginchar("0",ew#,eh#,0);
9.  z1=(0,0);
10.  z2=(w,0); z3=(w,h); z4=(0,h);
11.  z5=(.5w,a.top*h); z6=(.5w,a.bot*h);
12.  z7=whatever[z1,z5]=whatever[z4,z6];
13.  z8=whatever[z2,z5]=whatever[z3,z6];
14.  pickup pencircle scaled p;
15.  draw z1--z2--z3--z4--cycle; % kontur
16.  % klapka
17.  draw z4--z7{z7-z4}..{z3-z8}z8--z3;
18.  draw z1--z7; % lewa dolna przekątna
19.  draw z2--z8; % prawa przekątna
20.  labels(1,2,3,4,5,6,7,8);
21. endchar;
22.
23. end.
```

Linie 1–6 zawierają *parametry*: `golden_ratio`, `p#`, `size#`, `a.top`, `a.bot`, których konkretne wartości określają kształt koperty. Koperta będzie doskonała jeżeli długości jej boków pozostaną w złotej proporcji. Krótszy z boków prostokąta ma mieć długość `size#`, długość drugiego jest określona jako `size/golden_ratio`. Warto przyjrzeć się linii 4, w której w zwięzły sposób (typowy dla METAFONT-a) określono ten związek, i odpowiednie wartości przyporządkowano nowym zmiennym `eh#` i `ew#` (wysokość i długość koperty). Inną zasługującą na uwagę formułą jest czterokrotnie powtórzona `if unknown` *coś*`: ... fi`. W programie sztuczka ta jest użyta do sprawdzania czy odpowiednie zmienne zostały zdefiniowane i jeżeli nie zostały zdefiniowane to przyjmują one wartości standardowe. Tak więc jeżeli ktoś kiedyś będzie potrzebował np. koperty o wysokości 5 cm, wystarczy że utworzy pliczek `koperta5.mf` zawierający tylko dwie instrukcje:

```
size#:=5cm#;
input koperta;
```

Konstrukcja `if` *coś*`: ... fi` może być składnikiem instrukcji, o czym świadczy linia 3 (instrukcje są oddzielane średnikiem a w linii 3 go nie ma!). Jeżeli `p#` jest rzeczywiście nieznane to METAFONT przeczyta linie 3 i 4 jako: `20p# = golden_ratio * ew# = eh# = size#`. Parametr `p#` standardowo ma więc przyjąć wielkość równą jednej dwudziestej `size#`.



Rysunek 1.

Skromne słówko `mode_setup` z linii 7 jest makrodefinicją, i to makrodefinicją tak ważną, że występującą w każdym programie metafontowym. Tam ustalana jest m.in. wartość zmiennej `hppp` (horizontal pixels per point). Idea jest następująca: parametry programu metafontowego powinny być określone w jednostkach „rzeczywistych” takich

jak: punkty, milimetry, centymetry itd. W celu oznaczenia, że chodzi o wielkości rzeczywiste (Knuth nazywa je „sharp units") piszemy po nich znak #. Gdzieś podczas wykonywania programu powinno nastąpić przekształcenie: `coś:=coś# * hppp;`. Następna instrukcja w tej samej linii jest przykładem tego działania. Nadaje ona wartość parametrowi p# „odpowiedzialnemu" za szerokość piórka. Wartość „pixelowa" powstała przez pomnożenie rzeczywistej przez hppp zostaje następnie zaokrąglona. Innymi słowy p jest wielkością piórka mierzoną w pikselach urządzenia na które generujemy kopertę. Jeśli np. urządzeniem tym jest drukarka o rozdzielczości 300 DPI to p=7 ($p \cdot hppp = 7{,}47198$), dla fotonaświetlarki 1270 DPI, p=32 (31,63142) a dla drukarki o rozdzielczości 120 DPI p=3 (2,98882). Podane wartości odnoszą się do piórka o standardowej szerokości `1/20 size# = 20/36pt#`.

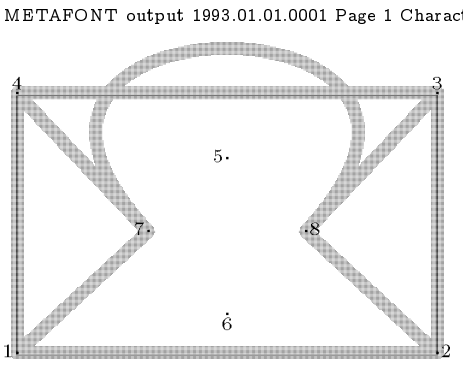
Makrodefinicja `beginchar` w linii 8 definiuje cztery rzeczy — kod znaku (tutaj ma on numer 48 — pozycja zera w kodzie ASCII), szerokość (width), wysokość (height) oraz głębokość (depth). Wszystkie te wielkości powinny być podane w jednostkach rzeczywistych ponieważ makro `beginchar` tworzy zmienne `w`, `h`, `d` i nadaje im odpowiednie wartości „pixelowe". W naszym przypadku 242, 149 i 0 pixeli.

Zawartość linii 9 i 10 nie jest żadną tajemnicą. Po prostu punkty z1, z2, z3, z4 są zdefiniowane jako leżące w czterech wierzchołkach prostokąta o długości boków `w` i `h`. W programie zastosowano najbardziej zwięzłą notację „z-ową", ale można je określać na kilka innych sposobów. Przykładowo punkt z1 z linii 9 możemy zdefiniować wykorzystując dowolny z poniższych zapisów:

```
(x1,y1)=(0,0)
x1=0; y1=0;
x1=y1=0;
```

Śledząc zapis w linii 11 dowiemy się po co były nam potrzebne parametry `a.top` i `a.bot`. Punkty 5 i 6 mają jednakże tylko charakter pomocniczy — posłużą do wyznaczenia dwóch brakujących dla wykreślenia koperty punktów.

Zapis `z7=whatever[z1,z5]` oznacza, że punkt z7 leży gdzieś na prostej określonej przez punkty z1 i z5. Ponieważ jednocześnie postulujemy, żeby tenże punkt z7 leżał na prostej `z1--z6`, jednoznacznie definiujemy go jako leżący na przecięciu tych prostych. Podobnie został określony punkt z8.



Rysunek 2.

Zaczynamy rysować kopertę. Najpierw w linii 14 za pomocą makra, `pickup pen` *jakiś* określamy kształt piórka. Nasze ma kształt kolisty, ale nie zawsze tak musi być. Inne możliwości to na przykład `pensquare` (kwadrat, jak sama nazwa wskazuje) lub `penrazor` (piórko o szerokości jednego pixela i zerowej wysokości). Słówko `scaled` to jedna z możliwych transformacji jakimi możemy poddać piórko. `pencircle scaled p` oznacza użycie pióra o okrągłym końcu o średnicy p#. Skalowanie można także przeprowadzić „w poziomie" instrukcją `xscaled` lub „w pionie" używając `yscaled`. Gdyby potrzebne nam było pióro o eliptycznym końcu, moglibyśmy je wybrać przez:

```
pickup pencircle xscaled px yscaled py
```

W liniach 15–19 rysujemy odpowiednie fragmenty koperty (znak procentu podobnie jak w składni TEX-owej rozpoczyna komentarz). Słowo `cycle` wskazuje, iż narysowana krzywa będzie się kończyć w punkcie z którego wyruszyła i gwarantuje, że w punkcie tym krzywa przebiegać będzie gładko. Zapis `z7{z7-z4}..{z3-z8}z8` oznacza, że życzymy sobie aby METAFONT rysował linię z punktu z7 w kierunku określonym przez wektor z7-z4, a z punktu z8 w kierunku określonym przez wektor z3-z8 (zapis `z7{z4-z7}..{z8-z3}z8` to nie to samo — patrz rysunek 2). Łącząc punkty w instrukcji `draw` znakiem `--` powodujemy wykreślanie odcinków, `..` spowoduje wykreślanie krzywych. W programie `koperta` taka linia użyta jest tylko raz — do wykreślenia klapki. Zwróćmy uwagę, że oprócz dwóch punktów startowych i podania dwóch kierunków nie ingerowaliśmy w sposób kreślenia tej linii. METAFONT zrobił to za nas i zrobił to tak, że poprawienie go nie wydaje się łatwym zadaniem.

Słowo endchar kończy definiowanie znaku rozpoczęte makrem beginchar. Słowo end. kończy pracę METAFONT-a, jeżeli go nie napiszemy METAFONT nie zakończy pracy i zgłosi gotowość dalszego działania wyświetlając gwiazdkę (dokładnie tak samo zachowuje się TEX). Znaczenie makrodefinicji label wyjaśni się za chwilę.

Teraz potrzebujemy programu METAFONT.EXE oraz paru dodatkowych plików. Wszystko co będzie nam potrzebne znaleźć możemy np. w pakiecie emTEX. Utwórzmy zatem katalog np. \EMTEX\EMMF i skopiujmy do niego następujące pliki: MF286.EXE lub MF.EXE (wersja dla komputerów z procesorem 8086), GFTOPK.EXE, GFTODVI.EXE. Ponadto do nowo utworzonego katalogu np. \EMTEX\MFBASES skopiujmy PLAIN.BAS.

Podobnie jak TEX, METAFONT po uruchomieniu musi odnaleźć i wczytać plik zawierający podstawowe makroinstrukcje (w TEX-u taki plik nazywa się formatem a w METAFON-cie bazą — np. PLAIN.BAS, lub CM.BAS, używana do generowania czcionek Computer Modern).

W opisie kodu generującego kopertę często używałem określeń: makro, makrodefinicja. Chodziło o konstrukcje znajdujące się właśnie w bazie. Baza, podobnie jak format TEX-owy, jest skompilowana, w celu przyśpieszenia jej czytania. W postaci źródłowej znajduje się ona w pliku PLAIN.MF. Tam można zobaczyć co znaczą: beginchar, mode_setup itd. Nie jest to jednak z pewnością lektura dla początkujących METAFONT-owców.

Używając pakietu emTEX powinniśmy określić zmienną środowiskową MFBAS. Podanie na przykład: SET MFBAS=C:\EMTEX\MFBASES, wskazuje, że bazy powinien METAFONT szukać w katalogu C:\EMTEX\MFBASES. Ponadto uruchamiając METAFONT-a powinniśmy określić na jakie urządzenie ma zostać wygenerowany font poprzez określenie odpowiedniej wartości zmiennej mode. Jeżeli generujemy kopertę na drukarkę laserową to możemy to zrobić tak:

```
mf286 \mode=hplaser; input koperta.mf
```

a dla drukarki mozaikowej epson-fx:

```
mf286 \mode=epsonfx; input koperta.mf
```

Otrzymamy po krótkiej chwili trzy pliki: koperta.tfm, koperta.300 (lub koperta.240 dla drukarki epson-fx), koperta.log. Co zawierają pliki z rozszerzeniami .tfm i .log łatwo się domyśleć, plik koperta.300 (koperta.240) zawiera obraz naszej koperty, ale w postaci niedostępnej dla sterowników np. pakietu emTEX. Potrzebny jest jeszcze jeden program — GFTOPK.EXE, przekształcający pliki typu .GF do postaci .PK. Piszemy po prostu:

```
gftopk koperta.300
```

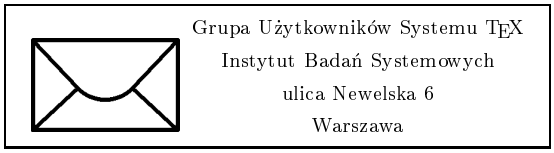
lub

```
gftopk koperta.240
```

I jeżeli program odpowie taką mniej więcej formułą:

```
This is GFtoPK, Version 2.2 [1g]
'METAFONT output 1993.01.01:0002'
516 bytes packed to 288 bytes.
```

oznacza to, że otrzymaliśmy plik koperta.pk, zawierający jeden znak, który jeżeli wszystko poszło dobrze powinien wyglądać mniej więcej tak:



Jeżeli koperta ma być drukowana na innej drukarce niż HP Laser lub Epson-FX musimy zadeklarować odpowiednią wartość zmiennej mode. W pakiecie E. Mattesa dostępne są następujące możliwości (są one określone w pliku LOCAL.MF): hplaser (HP LaserJet+), epsonfx (Epson FX-80), epsonmx (Epson MX-80), lqhires (NEC-P6 rozdzielczość 360×360), lqmedres (LQ-1500, NEC-P6 360×180), lqmedresl (LQ-1500, NEC-P6 180×360), lqlores (LQ-1500, NEC-P6 180×180), itoh (C.ITOH 8510A rozdzielczość 160×144). kyocera (Kyocera F-1010),

Wreszcie powinniśmy mieć możliwość obejrzenia znaków na etapie ich projektowania. Służy do tego program GFTODVI.EXE, generujący powiększony znak wraz z zaznaczonymi punktami kontrolnymi. Program ten tworzy obraz znaku posługując się specjalnymi czcionkami: grayfont i slantfont. Zestaw tych czcionek jest dostarczony w pakiecie EMTEX, ale w postaci plików źródłowych: gray.mf, slant.mf, slantlj.mf, graylj.mf, grayfx.mf oraz slantfx.mf. Dla drukarki mozaikowej generowanie tych fontów wygląda następująco (dla laserowej używamy plików graylj i slantlj oraz nadajemy odpowiednią wartość zmiennej mode):

```
mf286 \mode=epsonfx; input slantfx.mf
```

a potem

```
gftopk slantfx.240
```

i już mamy font slantfx. Pliki slantfx.tfm oraz slantfx.pk kopiujemy do odpowiednich katalogów, z których korzysta nasz TEX. W analogiczny sposób generujemy font grayfx, po czym uruchomiamy METAFONT-a po raz trzeci, tym razem znowu z plikiem koperta.mf, ale bez nadawania jakiejkolwiek wartości zmiennej mode (co nie oznacza, że zmienna ta jest niezdefiniowana — if unknown mode ...):

```
mf286 \input koperta.mf
lub*
mf286 \nodisplays; \input koperta.mf
```

Otrzymamy plik koperta.260, z którego generujemy plik koperta.dvi pisząc**:

```
gftodvi koperta.260/ grayfont graylj
  slantfont slantlj labelfont plr8
  titlefont plr10 /
```

Uwaga: aby uniknąć kłopotów i mało sensownych komunikatów o błędach powinniśmy dokładnie odtworzyć powyższą linię. W szczególności po nazwie pliku podanej koniecznie z rozszerzeniem nie ma odstępu, a po znaku / i przed kończącym / jest! W wyniku działania programu otrzymamy plik koperta.dvi zawierający jedną stronę, a na niej znajomy rysunek. Oglądamy go na ekranie lub drukujemy tak jak każdy inny plik DVI. Szary kontur koperty został złożony czcionkami grayfont i slantfont. Słowo METAFONT za pomocą czcionki logo8, oznaczenia punktów zadeklarowanych uprzednio instrukcją label czcionką labelfont (plr8) a data i godzina titlefont (plr10). Wygenerowany plik jest podstawą do oceny wyglądu znaku i usuwania ewentualnych błędów, na przykład takich jak na rysunku 2 (Co tam się mogło stać?).

W ten sposób instalcja METAFONT-a została zakończona. Pozostaje tylko usprawnienie wywoływania poszczególnych programów, aby zabawa była rzeczywiście zabawą. Ja wykorzystuję prosty plik M.BAT, kolejno wywołujący MF.EXE, GFTODVI.EXE, GFTOPK.EXE, i edytor. Każdy z nas może wykonać taki plik samodzielnie, kierując się własnymi przyzwyczajeniami.

Na koniec pytanie. Jaki znak powstanie w wyniku uruchomienia następującego programu? (Podpowiedź: także symbol z kolekcji „pocztowo-telekomunikacyjnej").

```
% Co to jest?

ht#:=18pt#;
mode_setup; define_pixels(ht);
u:=1/50ht; % jednostka

beginchar(0,18pt#,12pt#,0);
x0=.5w;y0=0;
y1'=y1=0; x1+16.5u=x1'-16.5u=x0;
y2=y2'=5u;x2+17u=x2'-17u=x0;
y3=y3'=18u;x3+12u=x3'-12u=x0;
y4=y4'=22u;x4+9u=x4'-9u=x0;
y5=y5'=23u;x5+6u=x5'-6u=x0;
y6=y6'=25u;x6+9u=x6'-9u=x0;
y7=y7'=25.5u;x7+7u=x7'-7u=x0;
x8=x9+1u;x8'-x0=x0-x8;y8=y8'=y9+2u;
x9=x0-13u;x9'-x0=x0-x9;
y9=y9'=y10=y10'=y4;
x10=x9-12u;x10'-x0=x0-x10;
x11'-x0=x0-x11;x11=x10+2.5u;
y11=y11'=y10+7u;
x99=x0;y99=13u;% środek tarczy
x88=x0;y88=h;
x20=x10;x19=x9;y20=y19=y10-1u;
x21=.5[x19,x20];y21=y20-2.5u;
x20'=x10';x19'=x9';y20'=y19'=y10'-1u;
x21'=.5[x19',x20'];y21'=y20'-2.5u;

filldraw z1--z1'--z2'--z3'--z4'--
  z6'--z7'--z5'--z5--z7--
  z6--z4--z3--z2--cycle;
filldraw z6..z8..z9--z10{up}..
  z11..tension1.44..z88..
  tension1.44..z11'..{down}
  z10'--z9'..z8'..z6'..tension1.55
  ..cycle; % rączka
filldraw z20{down}..{right}
  z21{right}..{up}z19--cycle;
filldraw z20'{down}..{left}
  z21'{left}..{up}z19'--cycle;

erase filldraw fullcircle
  scaled 14u shifted z99;
labels(range 0 thru 99);
endchar;
bye
```

*: Posiadacze karty EGA lub lepszej mogą wypróbować obie postacie instrukcji.

**: Na monitorze bez problemu całość zmieści się w jednej linii.

☎

Informacja

Wyciągi z protokółów

Grażyna Nowak

Warszawa, dn. 1992–01–08

Wyciąg z protokołu spotkania założycielskiego Polskiej Grupy Użytkowników Systemu TEX dotyczący wyboru Komitetu Założycielskiego.

Spotkanie założycielskie Polskiej Grupy Użytkowników Systemu TEX miało miejsce w dn. 8 stycznia 1992 roku w Warszawie. Obecnych było 39 osób.
Spotkanie rozpoczęło się od wyboru przewodniczącego zebrania. Jedynym zgłoszonym kandydatem była Hanna Kołodziejska. Została ona wybrana jednogłośnie.
Do Komitetu Założycielskiego zostały zgłoszone kolejno następujące osoby:

1. Hanna Kołodziejska
2. Janusz Sosnowski
3. Marek Ryćko
4. Anna Rudnik
5. Krzysztof Leszczyński

Został zgłoszony, a następnie przegłosowany wniosek o zamknięcie listy:

głosów *za*: 33
głosów *przeciw*: 2
głosów *wstrzymujących się*: 4

Kandydaci krótko przedstawili się wszystkim zebranym.
Został zgłoszony, a następnie przegłosowany wniosek o głosowanie na całą listę kandydatów:

głosów *za*: 31
głosów *przeciw*: 4
głosów *wstrzymujących się*: 4

Głosowanie całej listy:

głosów *za*: 33
głosów *przeciw*: 0
głosów *wstrzymujących się*: 6

Protokołowała: Anna Sencerz
Przewodniczący zebrania: Hanna Kołodziejska

Warszawa, dn. 1992–02–19

Wyciąg z protokołu spotkania założycielskiego Polskiej Grupy Użytkowników Systemu TEX GUST dotyczący uchwalenia statutu.

Spotkanie założycielskie Polskiej Grupy Użytkowników Systemu TEX miało miejsce w dn. 8 stycznia 1992 roku w Warszawie. Obecnych było 39 osób.
Podczas zebrania przedyskutowano statut stowarzyszenia.
Statut został przyjęty:

głosów *za*: 36
głosów *przeciw*: 0
głosów *wstrzymujących się*: 3

Protokołowała: Anna Sencerz
Przewodniczący zebrania: Hanna Kołodziejska

Warszawa, dn. 1992–06–05

Wyciąg z protokołu I Walnego Zebrania Polskiej Grupy Użytkowników Systemu TEX GUST

Zebranie odbyło się w Centrum Astronomicznym im. Mikołaja Kopernika w Warszawie, ul. Bartycka 18.
[...]

2. Wybór przewodniczącego Zebrania.

Ustalono, że na sali jest 20 spośród 39 członków założycieli, co stanowi wymagane w statucie quorum (1/3 ogólnej liczby członków). Spośród nich wybrano Janusza Sosnowskiego w głosowaniu jawnym przez podniesienie ręki na Przewodniczącego Zebrania (za — 18 głosów, przeciw — 0, wstrzymało się — 2).

3. Wybór Komisji Skrutacyjnej.

Do pracy w Komisji Skrutacyjnej zaproponowane zostały przez Przewodniczącego panie:

— Anna Rudnik
— Joanna Mickiewicz

W głosowaniu jawnym kandydatki zostały przyjęte większością głosów przy 1 głosie wstrzymującym się.

4. Przedstawienie propozycji dwustopniowych wyborów do Zarządu.

Przewodniczący Zebrania przedstawił konsultowaną z prawnikami propozycję przeprowadzenia wyborów

do Zarządu w dwóch turach. Zaproponował wybór Prezesa i 3 członków Zarządu spośród członków założycieli, następnie przerwę w Zebraniu w celu ukonstytuowania się Zarządu i przyjęcia nowych członków. Po przerwie — wybory uzupełniające do Zarządu. Propozycja ta została przedstawiona jako wniosek formalny i przyjęta w głosowaniu jawnym przy 1 głosie wstrzymującym się.
[...]

7. Wybór Prezesa.

Przewodniczący zapoznał zebranych z częścią Statutu mówiącą o obowiązkach Prezesa i Zarządu. Na stanowisko Prezesa kandydowali:

1. Włodzimierz Bzyl
2. Hanna Kołodziejska

Głosowanie było tajne, głosowało 19 osób, głosów ważnych było 19, 1 osoba wstrzymała się, pozostałe głosy rozdzieliły się następująco:

1. Włodzimierz Bzyl — 5 głosów
2. Hanna Kołodziejska — 13 głosów

Oznacza to, że Prezesem Grupy została p. H. Kołodziejska.

8. Wybór 3 członków Zarządu.

W tajnym głosowaniu oddano 19 głosów, wszystkie były ważne. Kandydatami były poniżej wypisane osoby i każda z nich otrzymała następującą liczbę głosów:

1. Włodzimierz Bzyl — 10 gł.
2. Krzysztof Leszczyński — 13 gł.
3. Włodzimierz Martin — 6 gł.
4. Piotr Pianowski — 8 gł.
5. Marek Ryćko — 10 gł.
6. Janusz Sosnowski — 8 gł.

W wyniku głosowania do Zarządu weszli:

1. Krzysztof Leszczyński — 13 gł.
2. Włodzimierz Bzyl — 10 gł.
3. Marek Ryćko — 10 gł.

9. Przerwa w Zebraniu.

Przewodniczący Zebrania zarządził przerwę, podczas której nowo wybrany Zarząd ukonstytuował się i przyjął w poczet członków osoby uczestniczące w zebraniu spoza członków założycieli, które złożyły deklaracje członkowskie.

10. Wybory uzupełniające do Zarządu.

Po przerwie M. Ryćko odczytał nazwiska nowo przyjętych 42 członków „GUST-u". (Od tej pory wymagane quorum wynosi 81/3, czyli 27 osób).
W głosowaniu jawnym, przy 2 głosach wstrzymujących się, ustalono liczbę osób w Zarządzie na 7 (plus Prezes). Na 4 vacaty kandydowali:

1. Marek Kowalówka (Katowice)
2. Jerzy Ludwichowski (Toruń)
3. Włodzimierz J. Martin (Gdańsk)
4. Stefan Paszkowski (Wrocław)
5. Piotr Pianowski (Sopot)
6. Janusz Sosnowski (Warszawa)
7. Joanna Tomasik-Krawczyk (Gliwice)

W głosowaniu tajnym głosów ważnych oddano 53, nieważny był 1. Głosy rozdzielono następująco:

1. Stefan Paszkowski (Wrocław) — 36 gł.
2. Janusz Sosnowski (Warszawa) — 34 gł.
3. Jerzy Ludwichowski (Toruń) — 32 gł.
4. Marek Kowalówka (Katowice) — 26 gł.
5. Włodzimierz J. Martin (Gdańsk) — 26 gł.
6. Piotr Pianowski (Sopot) — 24 gł.
7. Joanna Tomasik-Krawczyk (Gliwice) — 24 gł.

W związku z faktem, iż pp. M. Kowalówka oraz Wł. J. Martin uzyskali tę samą liczbę głosów, Przewodniczący postawił wniosek o rozszerzenie składu Zarządu do 8 osób. W głosowaniu jawnym przy 2 głosach przeciw i 5 wstrzymujących się wniosek został przyjęty.
Podsumowując wyniki wyborów w skład Zarządu weszli:

1. Hanna Kołodziejska (Warszawa)–PREZES
2. Włodzimierz Bzyl (Gdańsk)
3. Marek Kowalówka (Katowice)
4. Krzysztof Leszczyński (Warszawa)
5. Jerzy Ludwichowski (Toruń)
6. Włodzimierz J. Martin (Gdańsk)
7. Stefan Paszkowski (Wrocław)
8. Marek Ryćko (Warszawa)
9. Janusz Sosnowski (Warszawa)

[...]

Protokołowała: Grażyna Nowak
Przewodniczący zebrania: Janusz Sosnowski

LaTeX

TeX na goownicy*

Hanna Kołodziejska

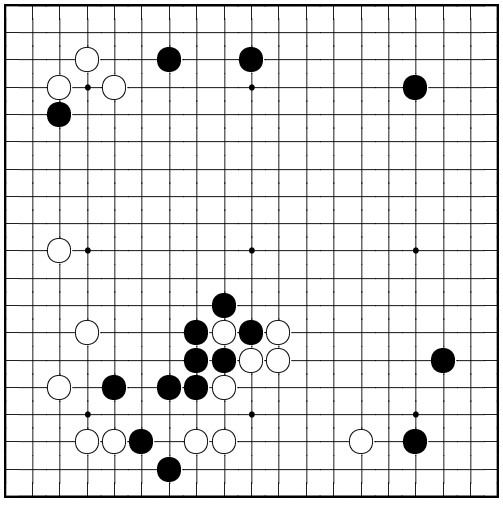
1 Wprowadzenie

Grą w Go zainteresowałam się kilka lat temu dzięki kolegom z Wydziału Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego. Usiłując zgłębić tajniki tej pasjonującej gry zaczęłam czytać literaturę goistyczną. Niewielki był pożytek z mojego czytania, gdyż bardziej niż strategia gry zaciekawił mnie problem składania TeX-em diagramów. Ponadto w owym czasie trafiłam na artykuł Zalmana Rubinsteina o składaniu TeX-em diagramów szachowych [1]. I tak powstały moje fonty i makra do Go.

2 Co to jest Go

Go jest grą planszową, znaną od czterech tysięcy lat. Pochodzi z Chin, skąd poprzez Koreę i Japonię rozprzestrzeniło się na cały świat.

Planszę do Go stanowi drewniana deska z zaznaczonymi 19 poziomymi i 19 pionowymi liniami. Piony o kształcie soczewek nazywa się kamieniami, nawet gdy są wykonane z plastyku lub szkła. Podczas gry są one umieszczane na przecięciach linii (rys. 1).

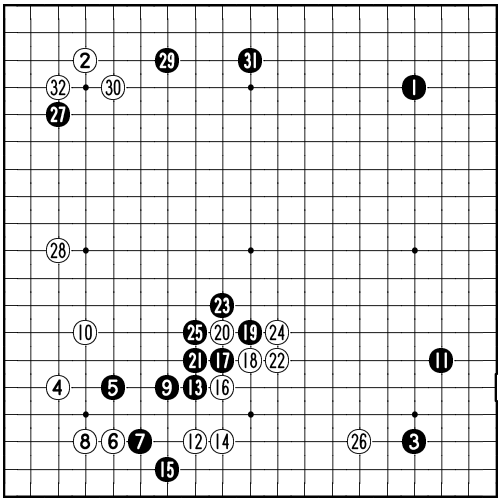


Rys. 1

* Jest to oczywiście nazwa planszy do Go.

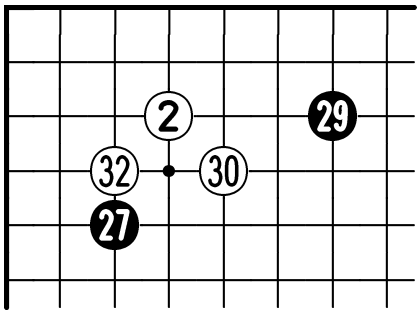
Na początku gry plansza jest pusta. Gracze na zmianę kładą na niej swoje piony, po jednym w każdym ruchu. Jeden z graczy gra czarnymi pionami, drugi białymi. Raz położony pion nie zmienia swojej pozycji (ewentualnie może być zdjęty z planszy w wyniku zbicia).

Kolejność ruchów zaznacza się na diagramach numerując piony: ❶ oznacza pierwszy pion w grze, ② — drugi itd. Przykładowa sytuacja na planszy z zaznaczoną kolejnością ruchów jest przedstawiona na rys. 2.



Rys. 2

Często zachodzi także potrzeba pokazania tylko pewnego fragmentu planszy. Może to być szczególna sytuacja podczas gry lub niezależny problem. Na rysunku 3 jest przedstawiony fragment planszy z rys. 2.

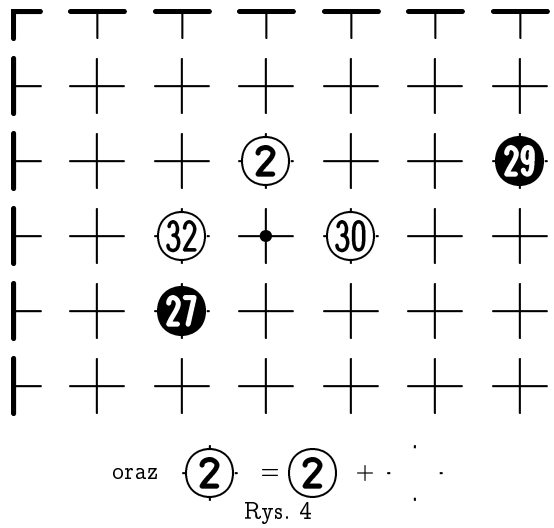


Rys. 3

3 Diagramy TEX-owe — definiowanie znaków w Metafoncie

Przystępując do pracy nad diagramami do Go zdecydowałam się zdefiniować i wygenerować przy pomocy METAFONTa wszystkie potrzebne symbole, nawet linie i kółka. W rezultacie sposób wstawiania diagramów do tekstu TEX-owego nie zależy od tego, czy pracujemy z plain TEX-em czy z też LATEX-em.

Przyjrzyjmy się, z jakich symboli składa się diagram do Go. Weźmy, na przykład, diagram z rys. 3 i rozłóżmy go na symbole składowe. Rezultat jest przedstawiony na rys. 4.



Rys. 4

Po przyjrzeniu się diagramowi rozłożonemu na proste elementy, możemy stwierdzić, że do składania diagramów TEX-em potrzebne są następujące znaki:

- białe piony: ◯
  - z numerami: ① ② ③ itd.
  - z dodatkowymi symbolami: △ □
- czarne piony: ●
  - z numerami: ❶ ❷ ❸ itd.
  - z dodatkowymi symbolami: ▲ ■
- przecięcia linii: ┼ ┼
  - linie boczne: ├ └ itd.
  - „uzupełnienia”: · ·
  - „uzupełnienia” linii bocznych: · · · itd.

Skoncentrujmy się na pionach białych. W przypadku czarnych rozumowanie oraz postępowanie jest identyczne. Przyjrzyjmy się następującym pionom:

① ② ⑫ ⑫

Zauważmy różnicę grubości i kształtu cyfr na kolejnych pionach. Moje podejście było następujące:

1. Wygenerowałam znak ◯ i zapamiętałam go pod nazwą `white_circle`.
2. Napisałam dziesięć METAFONT-owych makrodefinicji opisujących kształt cyfr: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, po czym zapamiętałam je również jako obiekty typu `picture`.
3. Za pomocą parametrów zmieniałam kształt cyfr oraz ich proporcje.

Oto moje makro do utworzenia znaku ◯:

```
beginchar(0,24/22size#,23/22size#,1/22size#);
 pickup pencircle scaled line_thickness;
 lft x1=1/24w; y1=11/24w;
 rt x2=23/24w; y2=11/24w;
 draw z1..z2..cycle;
 white_circle := currentpicture;
showit; endchar;
```

Makro dla cyfry 1:

```
def digit_one =
 currentpicture := nullpicture;
 pickup pencircle scaled dig_pen;
 x1=.5dig_w; bot y1=0;
 x2=.5dig_w; top y2=dig_h;
 draw z1--z2;
 digit[1] := currentpicture;
 clearxy; clearit; clearpen;
enddef;
```

Fragment kodu w języku METAFONT dla znaku ①:

```
beginchar(1,24/22size#,23/22size#,1/22size#);
 currentpicture := white_circle
  + digit[1] shifted
     (.5w-.5dig_w,11/24w-.5dig_h);
showit;
endchar;
```

Spróbujmy teraz analogicznie otrzymać znak ⑫. Najpierw należy zmienić wartości parametrów określających szerokość cyfry i szerokość pióra (ang. *pen*) oraz wygenerować nowe obiekty typu `picture`.

```
 dig_w := .7dig_w;
 dig_pen := .7dig_pen;
 digit_one; digit_two;
```

Znak ⑫ możemy zdefiniować następująco:

```
beginchar(12,24/22size#,23/22size#,1/22size#);
 currentpicture := white_circle
  + digit[1] shifted
      (.5w-1.2dig_w,11/24w-.5dig_h)
  + digit[2] shifted
      (.5w,11/24w-.5dig_h);
showit;
endchar;
```

Po wygenerowaniu wszystkich potrzebnych do Go znaków otrzymałam fonty trzech typów:

1. fonty zawierające czarne piony, na przykład gobla10 (czarne piony wielkości 10pt);
2. fonty zawierające białe piony, na przykład gowhi10 (białe piony wielkości 10pt);
3. fonty zawierające dodatkowe symbole (przecięcia linii, linie brzegowe itp.), np. go10 (wielkość 10pt).

Prawdopodobnie kolejne fonty z pionami o numeracji powyżej 255 mogą okazać się potrzebne, ponieważ nie należą do rzadkości dłużej trwające gry!

4 Makrodefinicje TEX-owe

Makrodefinicje potrzebne do kodowania diagramów do Go znajdują się w pliku go.sty. Jak wspomniałam, można ich używać zarówno w plain TEX-u jak i w LATEX-u. Dołączenie pliku go.sty do składanego tekstu odbywa się komendą \input lub dodatkową opcją go LATEX-owej komendy \documentstyle.

W literaturze do Go nie ma zwyczaju (o ile mi wiadomo) specjalnego oznaczania przecięć linii na planszy, czyli miejsc, na których są umieszczane piony. Musiałam jednak wprowadzić oznaczenia, aby w wywołaniach makroinstrukcji TEX-owych dokładnie określić pozycję na planszy. W moich makrach każde przecięcie linii jest określone przez numer linii poziomej czyli *wiersza* (jedna z liter: a, b, c, d, e, f, g, h, i, k, l, m, n, o, p, q, r, s, t) i numer linii pionowej czyli *kolumny* (liczba od 1 do 19).

Zdefiniowałam następujące komendy TEX-owe wystarczające do składania diagramów do Go:

- wybór wielkości pionów i innych elementów diagramu (w punktach):

  ```
  \gofontsize{10}
  ```

  domyślnie: 10pt
- inicjalizacja całej planszy:

  ```
  \inifulldiagram lub
  \inidiagram␣a-t:1-19␣
  ```

  (konieczna spacja ogranicza czwarty parametr)
- inicjalizacja fragmentu planszy:

  ```
  \inidiagram␣a-t:1-19␣
  ```

  (konieczna spacja ogranicza czwarty parametr) przy czym dwa pierwsze parametry określają wiersze, a dwa kolejne kolumny (od–do)
- określenie piona:

  ```
  \black.              \white.
  \black{1}            \white{2}
  \black{\square}      \white{\triangle}
  ```

  przy czym kropka oznacza, że dany pion nie jest numerowany.
- inne określenie pozycji:

  ```
  \letter{a}
  \symbol{?}
  ```
- postawienie piona (lub zaznaczenie ruchu):

  ```
  \pos{a}{5}=\black{1}
  \pos{a}{5}=\letter{a}
  ```
- pokazanie sytuacji na planszy:

  ```
  \showfulldiagram  lub
  \showdiagram␣a-t:1-19␣
  ```

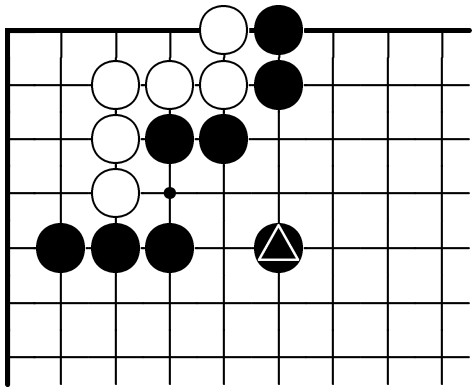
  (konieczna spacja ogranicza czwarty parametr)
- pokazanie sytuacji na części planszy:

  ```
  \showdiagram␣a-t:1-19␣
  ```

  (konieczna spacja ogranicza czwarty parametr) przy czym dwa pierwsze parametry określają wiersze, a dwa kolejne kolumny (od–do)

5 Przykład

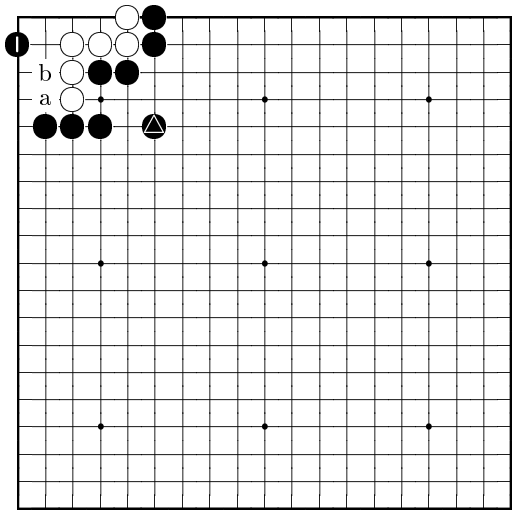
Na rysunku 5 został przedstawiony problem, a jego rozwiązanie na rys. 6.



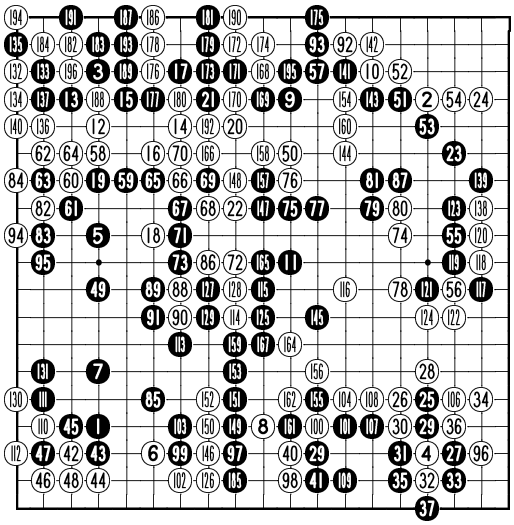
Rys. 5

```
\input go.sty %wczytanie pliku z makrami
\gofontsize{20} % wielkość pionów/symboli
% (domyślnie 10pt)
\pos{a}{5}=\white.
% położenie białego piona na A5
% (bez żadnego numeru)
\pos{a}{6}=\black.
% położenie czarnego piona na A6
% (bez żadnego numeru)
```

```
\pos{b}{3}=\white.
\pos{b}{4}=\white.
\pos{b}{5}=\white.
\pos{b}{6}=\black.
\pos{c}{3}=\white.
\pos{c}{4}=\black.
\pos{c}{5}=\black.
\pos{d}{3}=\white.
\pos{e}{2}=\black.
\pos{e}{3}=\black.
\pos{e}{4}=\black.
\pos{e}{6}=\black{\triangle}
% położenie czarnego piona
% oznaczonego trójkątem
$$
\showdiagram a-g:1-9
% wynik pokazany na rys. 5
$$
\pos{b}{1}=\black{1}
% położenie czarnego piona z 1
\pos{c}{2}=\letter{b}
% wpisanie litery ,,b'' na C2
\pos{d}{2}=\letter{a}
\gofontsize{10}
% zmiana wielkości pionów
% i innych symboli
$$
\showfulldiagram
% jak na rys. 6
$$
\inifulldiagram
% ponowna inicjalizacja planszy
```



Rys. 6



Rys. 7

Przykład prawdziwej gry przedstawia rys. 7.

6 TEX-owe makra od środka

Moim zasadniczym pomysłem było reprezentowanie każdego przecięcia linii na planszy przez TEX-ową komendę następującej postaci:

\@<*litera*><*cyfra rzymska*>

przy czym *litera* i *cyfra rzymska* określają odpowiednio wiersz i kolumnę. Oznacza to reprezentowanie planszy do Go jako tablicy 19×19 i pamiętanie wartości każdego jej elementu:

```
\@ai  \@aii  \@aiii  \@aiv  ...  \@axix
\@bi  \@bii  \@biii  \@biv  ...  \@bxix
 ..................................
\@ti  \@tii  \@tiii  \@tiv  ...  \@txix
```

Makrodefinicja \pos, o podstawowym znaczeniu dla składania diagramów, została zdefiniowana w następujący sposób:

```
\def\pos\#1\#2=\#3\#4{ .......
 \expandafter\let
 \csname@\#1\romannumeral\#2\endcsname=0
 \relax
 \edef\@pos{
  \def\csname@\#1\romannumeral\#2
       \endcsname{\#3{\@fourth}}}\@pos
 \ignorespaces}
```

Komenda \edef (*expanded definition*) jest używana także w innych makrach do składania diagramów do Go.

7 Uwagi końcowe

Diagramy są włączane do tekstu jako zwykłe pionowe pudełka (*vboxes*).

Piony mogą być także wstawiane bezpośrednio w tekst akapitu. Wówczas wystarczy użyć komend `\textwhite` i `\textblack` zamiast `\white` i `\black`. Na przykład zdanie z początku tego artykułu: „❶ oznacza pierwszy pion w grze, ② — drugi" zostało złożone następująco: „`\textblack{1}` oznacza pierwszy pion w grze, `\textwhite{2}` — drugi".

Nie ma żadnych innych tajemnic w składaniu TEX-em diagramów do Go. Kształt cyfr użytych do numeracji pionów w grze jest daleki od doskonałości. Może ktoś podjąłby się utworzenia lepszych fontów?

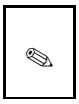
Makrodefinicje TEX-owe przeznaczone do składania diagramów 9×9 lub 13×13 można bez kłopotów dodać do pliku `go.sty` modyfikując istniejące makra dla diagramów 19×19.

Literatura

[1] Zalman Rubinstein: *Chess printing via Metafont and TEX*. TUGboat Vol. 10, No. 2.

[2] Iwamoto Kaoru, 9 dan: *Go for beginners*. Ishi Press, Inc., Tokyo 1972.

[3] Janusz Kraszek: *Świat Go*. COK, Warszawa 1989.



Uciecha z TEX-a

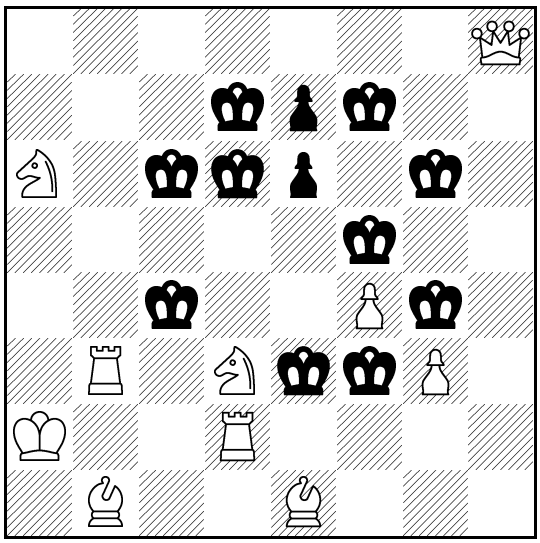
Krzyżowka

W wolne pola wpisz komendy TEXa.

\		G			
\		U			
\		S			
\		T			

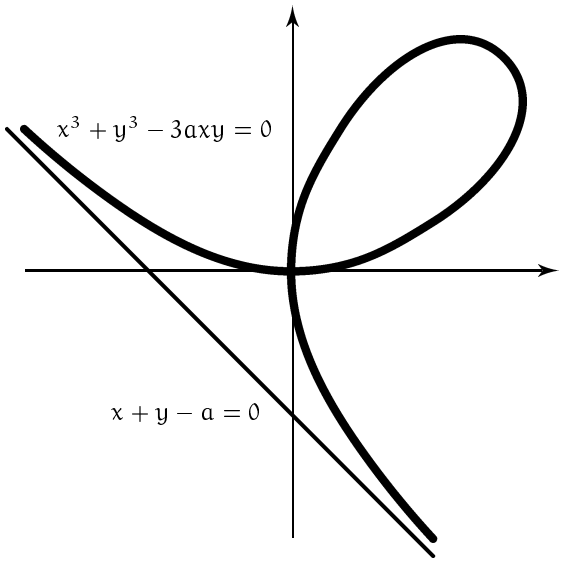
Jak nazywa się ten font?

Szachy



Białe zaczynają i wygrywają

Wykres



Plik go.sty

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                       %
%    This is `go.sty' ver. 0.07 (September 1992)                        %
%    (c) Copyright by Hanna Kołodziejska, 1992                          %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\catcode`@=11

\newbox\@gobox \newdimen\@godimen

\def\gofontsize#1{
    \font\gofont=go#1 at #1truept
    \font\blackfont=gobla#1 at #1truept
    \font\whitefont=gowhi#1 at #1truept
    \ifnum #1=10 \font\letterfont=cmr10 at 10truept\else        %% <--- cm
        \ifnum #1=15 \font\letterfont=cmr10 at 14.4truept\else
        \ifnum #1=20 \font\letterfont=cmr10 at 17.28truept\fi\fi\fi
    \setbox\@gobox=\hbox{\gofont\char0}
    \@godimen=\wd\@gobox
    }

\gofontsize{10} % initialization <---

\def\newgoline{\hfill\break}
\def\hoshi{\gofont\char0}
\def\empty{\gofont\char1}
\def\lftborder{\gofont\char2}
\def\rtborder{\gofont\char3}
\def\topborder{\gofont\char4}
\def\botborder{\gofont\char5}
\def\lfttopcorner{\gofont\char6}
\def\rttopcorner{\gofont\char7}
\def\lftbotcorner{\gofont\char8}
\def\rtbotcorner{\gofont\char9}
\def\triangle{\whitefont\char255}
\def\square{\whitefont\char254}

\newcount\n
\newcount\@beglet \newcount\@endlet
\newcount\@lettercount

\def\@letternumber#1{\relax
    \ifx #1a\@lettercount=1\else
    \ifx #1b\@lettercount=2\else
    \ifx #1c\@lettercount=3\else
    \ifx #1d\@lettercount=4\else
    \ifx #1e\@lettercount=5\else
    \ifx #1f\@lettercount=6\else
    \ifx #1g\@lettercount=7\else
    \ifx #1h\@lettercount=8\else
    \ifx #1i\@lettercount=9\else
    \ifx #1k\@lettercount=10\else
    \ifx #1l\@lettercount=11\else
```

```
    \ifx #1m\@lettercount=12\else
    \ifx #1n\@lettercount=13\else
    \ifx #1o\@lettercount=14\else
    \ifx #1p\@lettercount=15\else
    \ifx #1q\@lettercount=16\else
    \ifx #1r\@lettercount=17\else
    \ifx #1s\@lettercount=18\else
    \ifx #1t\@lettercount=19\else
       \errmessage{Row label must be letter!}
    \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
    \fi\fi\fi\fi\fi\fi\fi\fi\fi
    }

\def\inidiagram#1-#2:#3-#4 {\relax
    \ifnum #3>#4 \errmessage{Invalid column numbers!} \fi
    \@letternumber{#1} \@beglet=\@lettercount
    \@letternumber{#2} \@endlet=\@lettercount
    \def\@inirow##1##2{{\n=#3
      \loop
       \expandafter\let\csname @##1\romannumeral\n \endcsname=0\relax
       \edef\@@inirow{\global
            \def\csname @##1\romannumeral\n \endcsname{##2}}\@@inirow
       \ifnum \n<#4 \advance \n by 1
      \repeat
      }}\relax
    \loop
      \ifcase\@beglet\relax\or
      \@inirow{a}{\topborder}\relax
        \ifnum #3=1 \global\def\@ai{\lfttopcorner}\fi
        \ifnum #4=19 \global\def\@axix{\rttopcorner}\fi \or
      \@inirow{b}{\empty}\relax
        \ifnum #3=1 \global\def\@bi{\lftborder}\fi
        \ifnum #4=19 \global\def\@bxix{\rtborder}\fi \or
      \@inirow{c}{\empty}\relax
        \ifnum #3=1 \global\def\@ci{\lftborder}\fi
        \ifnum #4=19 \global\def\@cxix{\rtborder}\fi \or
      \@inirow{d}{\empty}\relax
        \ifnum #3=1 \global\def\@di{\lftborder}\fi
        \ifnum #3<5  \ifnum #4>3  \global\def\@div{\hoshi}\fi\fi
        \ifnum #3<11 \ifnum #4>9  \global\def\@dx{\hoshi}\fi\fi
        \ifnum #3<17 \ifnum #4>15 \global\def\@dxvi{\hoshi}\fi\fi
        \ifnum #4=19 \global\def\@dxix{\rtborder}\fi \or
      \@inirow{e}{\empty}
        \ifnum #3=1 \global\def\@ei{\lftborder}\fi
        \ifnum #4=19 \global\def\@exix{\rtborder}\fi \or
      \@inirow{f}{\empty}
        \ifnum #3=1 \global\def\@fi{\lftborder}\fi
        \ifnum #4=19 \global\def\@fxix{\rtborder}\fi \or
      \@inirow{g}{\empty}
        \ifnum #3=1 \global\def\@gi{\lftborder}\fi
        \ifnum #4=19 \global\def\@gxix{\rtborder}\fi \or
      \@inirow{h}{\empty}
        \ifnum #3=1 \global\def\@hi{\lftborder}\fi
        \ifnum #4=19 \global\def\@hxix{\rtborder}\fi \or
      \@inirow{i}{\empty}
```

```
      \ifnum #3=1 \global\def\@ii{\lftborder}\fi
      \ifnum #4=19 \global\def\@ixix{\rtborder}\fi \or
    \@inirow{k}{\empty}
      \ifnum #3=1 \global\def\@ki{\lftborder}\fi
      \ifnum #3<5  \ifnum #4>3  \global\def\@kiv{\hoshi}\fi\fi
      \ifnum #3<11 \ifnum #4>9  \global\def\@kx{\hoshi}\fi\fi
      \ifnum #3<17 \ifnum #4>15 \global\def\@kxvi{\hoshi}\fi\fi
      \ifnum #4=19 \global\def\@kxix{\rtborder}\fi \or
    \@inirow{l}{\empty}
      \ifnum #3=1 \global\def\@li{\lftborder}\fi
      \ifnum #4=19 \global\def\@lxix{\rtborder}\fi \or
    \@inirow{m}{\empty}
      \ifnum #3=1 \global\def\@mi{\lftborder}\fi
      \ifnum #4=19 \global\def\@mxix{\rtborder}\fi \or
    \@inirow{n}{\empty}
      \ifnum #3=1 \global\def\@ni{\lftborder}\fi
      \ifnum #4=19 \global\def\@nxix{\rtborder}\fi \or
    \@inirow{o}{\empty}
      \ifnum #3=1 \global\def\@oi{\lftborder}\fi
      \ifnum #4=19 \global\def\@oxix{\rtborder}\fi \or
    \@inirow{p}{\empty}
      \ifnum #3=1 \global\def\@pi{\lftborder}\fi
      \ifnum #4=19 \global\def\@pxix{\rtborder}\fi \or
    \@inirow{q}{\empty}
      \ifnum #3=1 \global\def\@qi{\lftborder}\fi
      \ifnum #3<5  \ifnum #4>3  \global\def\@qiv{\hoshi}\fi\fi
      \ifnum #3<11 \ifnum #4>9  \global\def\@qx{\hoshi}\fi\fi
      \ifnum #3<17 \ifnum #4>15 \global\def\@qxvi{\hoshi}\fi\fi
      \ifnum #4=19 \global\def\@qxix{\rtborder}\fi \or
    \@inirow{r}{\empty}
      \ifnum #3=1 \global\def\@ri{\lftborder}\fi
      \ifnum #4=19 \global\def\@rxix{\rtborder}\fi \or
    \@inirow{s}{\empty}
      \ifnum #3=1 \global\def\@si{\lftborder}\fi
      \ifnum #4=19 \global\def\@sxix{\rtborder}\fi \or
    \@inirow{t}{\botborder}
      \ifnum #3=1 \global\def\@ti{\lftbotcorner}\fi
      \ifnum #4=19 \global\def\@txix{\rtbotcorner}\fi
    \fi
    \ifnum \@beglet<\@endlet \advance \@beglet by 1
  \repeat
  }

\def\inifulldiagram{\inidiagram a-t:1-19 }

\inifulldiagram % initialization

\def\showdiagram#1-#2:#3-#4 {\vbox{\offinterlineskip\noindent
    \ifnum #3>#4 \errmessage{Invalid column numbers!} \fi
    \hsize=\@godimen
    \n=#4\advance\n by-#3\advance\n by1
    \multiply\hsize by\n
    \@letternumber{#1} \@beglet=\@lettercount
    \@letternumber{#2} \@endlet=\@lettercount
    \def\@showrow##1{{\n=#3
```

```
      \loop
        \edef\@@showrow{\csname @##1\romannumeral\n \endcsname}\@@showrow
        \ifnum \n<#4 \advance \n by 1
      \repeat
      }}\relax
    \loop
      \ifcase\@beglet\relax\or
        \@showrow{a}\or
        \@showrow{b}\or
        \@showrow{c}\or
        \@showrow{d}\or
        \@showrow{e}\or
        \@showrow{f}\or
        \@showrow{g}\or
        \@showrow{h}\or
        \@showrow{i}\or
        \@showrow{k}\or
        \@showrow{l}\or
        \@showrow{m}\or
        \@showrow{n}\or
        \@showrow{o}\or
        \@showrow{p}\or
        \@showrow{q}\or
        \@showrow{r}\or
        \@showrow{s}\or
        \@showrow{t}
      \fi
      \ifnum \@beglet<\@endlet
        \newgoline \advance \@beglet by 1
    \repeat
    }}

\def\showfulldiagram{\showdiagram a-t:1-19 }

\def\pos#1#2=#3#4{\relax
    \ifcat #1a\relax \else \errmessage{Row label must be a letter!}\fi
    \ifnum #2<1 \errmessage{Column number less than 1?}\fi
    \ifnum #2>19 \errmessage{Column number greater than 19?}\fi
    \edef\@fourth{#4}\relax
    \ifx .#4\ifx #3\empty\edef\@fourth{}\else \edef\@fourth{0}\fi\fi
    \edef\@borders{\relax
        \ifx #1a \ifnum #2=1  \gofont\char15 \else
                 \ifnum #2=19 \gofont\char16 \else
                 \gofont\char13 \fi\fi
        \else
        \ifx #1t \ifnum #2=1  \gofont\char17 \else
                 \ifnum #2=19 \gofont\char18 \else
                 \gofont\char14 \fi\fi
        \else
        \ifnum #2=1 \gofont\char11
        \else
        \ifnum #2=19 \gofont\char12
        \else
        \gofont\char10
        \fi\fi\fi\fi}\relax
```

```
    \expandafter\let\csname @#1\romannumeral#2\endcsname=0\relax
    \edef\@pos{\def\csname @#1\romannumeral#2\endcsname{#3{\@fourth}}}\@pos
    \ignorespaces
    }

\newcount\nr

\def\black#1{\relax
    \ifx \triangle#1{\let\whitefont=\blackfont
                    \rlap{\triangle}\@borders}\else
    \ifx \square#1{\let\whitefont=\blackfont
                    \rlap{\square}\@borders}\else
    \ifnum #1<0 \errmessage{Negative Black's move?}\else
    \ifnum #1>253 \errmessage{Black's move too big!}\else
    {\blackfont\rlap{\char#1}\@borders}\ignorespaces
    \fi\fi\fi\fi}

\def\white#1{\relax
    \ifx \triangle#1{\rlap{\triangle}\@borders}\else
    \ifx \square#1{\rlap{\square}\@borders}\else
    \ifnum #1<0 \errmessage{Negative White's move?}\else
    \ifnum #1>253 \errmessage{White's move too big!}\else
    {\whitefont\rlap{\char#1}\@borders}\ignorespaces
    \fi\fi\fi\fi}

\def\textblack#1{\def\@borders{\hskip\wd\@gobox}\leavevmode
    \ifx .#1\lower.2\wd\@gobox\hbox{\black{0}}\else
      \lower.2\wd\@gobox\hbox{\black{#1}}\fi}

\def\textwhite#1{\def\@borders{\hskip\wd\@gobox}\leavevmode
    \ifx .#1\lower.2\wd\@gobox\hbox{\white{0}}\else
      \lower.2\wd\@gobox\hbox{\white{#1}}\fi}

\newbox\@letterbox \newdimen\@letterdim

\def\symbol#1{\relax
    \ifcat a#1\relax\else
    \ifcat 0#1\relax\else
       \errmessage{Strange parameter of symbol macro! (#1)}
    \fi\fi
    \setbox\@letterbox=\hbox to\wd\@gobox{\hfil\letterfont #1\hfil}\relax
    \@letterdim=\wd\@gobox
    \divide \@letterdim by 24
    \multiply \@letterdim by 22
    \advance \@letterdim by -\ht\@letterbox
    {\raise.5\@letterdim\box\@letterbox}\relax
    }

\let\letter=\symbol

\catcode`@=12
\endinput
```

✓ **MEX**

Polerowanie TEX-a: od systemu gotowego do użycia do systemu wygodnego w użyciu

Bogusław Jackowski i Marek Ryćko*

Abstract

W artykule opisana została polska adaptacja systemu składu TEX (dla wersji ⩾3.0). Autorzy starali się pokazać, że z profesjonalnego punktu widzenia TEX *nie jest systemem wielojęzycznym* i że przystosowanie go do konkretnego języka nie jest bynajmniej zadaniem trywialnym. W artykule omówione są subtelne aspekty takiego przystosowania, dotyczące zarówno TEX-owej, jak i METAFONT-owej części zadania. Pokrótce omówione jest również łącze TEX–PostScript.

Słowa kluczowe: wersje narodowe formatów `plain` i LATEX, znaki narodowe, podcięcia (*kerns*), ligatury, METAFONT, PostScript.

1 Wstęp

Podejmując pierwszą próbę przystosowania TEX-a do języka polskiego [Jackowski et al. 88] byliśmy — jak to widzimy z dzisiejszej perspektywy — bardzo naiwni. Rok poźniej prezentując pracę o szumnym tytule "Polish TEX is ready for use" (*Polski TEX jest gotowy do użycia*) nie spodziewaliśmy się, że czeka nas jeszcze tak długa droga do uzyskania adaptacji zadowalającej z profesjonalnego punktu widzenia. Czym się różni skład „profesjonalny" od „nieprofesjonalnego"? Naszym zdaniem skład „nieprofesjonalny" po prostu nie istnieje.

Po kilku latach pracy TEX wreszcie został przystosowany do składania tekstów w języku polskim. Staraliśmy się nie uronić żadnej z TEX-owych subtelności wierząc, że właśnie subtelności stanowią o wysokiej jakości systemu składu. Trzeba przyznać, że mieliśmy sporo szczęścia: nasz język okazał się przystosowywalny, co wcale nie było takie oczywiste.

Opracowane zostały — rzecz jasna — zasady dzielenia wyrazów w języku polskim, opracowane (i zaprogramowane w języku METAFONT) zostały też kształty polskich znaków diakrytycznych, ale to jeszcze nie wszystko. Użytkownik ma możliwość definiowania instrukcji zawierających w nazwie polskie litery, napisy pojawiające się na ekranie i tworzone w trakcie obróbki tekstu pliki mogą zawierać polskie znaki, o ile polskie znaki są dostępne na danej instalacji, a dana implementacja TEX-a ma konfigurowalną tabelę kodów (*code page*). Dopuszczalna jest również dwuznakowa notacja dla polskich znaków diakrytycznych. Polskie odpowiedniki formatów `plain` i LATEX są z tymi formatami w pełni zgodne w tym sensie, że polskie formaty stanowią rozszerzenie formatów oryginalnych. I tak dalej.

Tym niemniej nie jest nadal oczywiste, czy TEX daje się przystosować do dowolnego języka, a tym bardziej czy można utworzyć wielojęzyczną adaptację dla dowolnego zestawu języków. Nasze doświadczenia zdają się sugerować negatywną odpowiedź w obu przypadkach. O tych doświadczeniach traktuje właśnie nasza praca, a czytelnikowi pozostawiamy wyciągnięcie ostatecznych wniosków co do wielojęzyczności TEX-a.

Niezależnie od osądu czytelnika TEX jest dokładnie taki jaki jest. Niewątpliwie posiada pewne cechy wielojęzyczności, które można wykorzystywać w różny sposób. Chcielibyśmy wskazać czytelnikowi niebezpieczeństwa kryjące się za niektórymi rozwiązaniami.

Ograniczymy się do rozważenia jedynie dwóch zagadnień, mianowicie opiszemy polskie rozszerzenie fontów Computer Modern i przeanalizujemy problem zastosowania ligatur jako metody dostępu do znaków narodowych w foncie. Powody po temu są następujące: (1) w przeciwnym wypadku artykuł byłby stanowczo za długi; (2) szczegółowy opis tych dwóch zagadnień pozwala wyrobić sobie zdanie o całym procesie przystosowywania TEX-a do pracy w danym języku.

2 Fonty

Fonty Computer Modern (CM) nie nadają się do składania polskich tekstów. Głównym powodem jest brak w tych fontach liter 'ą', 'ę', 'Ą' i 'Ę'. Brak nawet znaku „ogonek", chociaż — jak okaże się dalej — nawet gdyby fonty CM taki znak zawierały, wiele by to

* Jest to tłumaczenie pracy „Polishing TEX: from ready to use to handy in use", która została wyróżniona nagrodą im. Cathy Booth na konferencji EuroTEX'92 w Pradze

nie pomogło. Tym samym byliśmy zmuszeni zastosować podejście odmienne od podejścia zastosowanego przez Holendrów w projekcie Babel czy Niemców w pakiecie LaTeX. Nie mogliśmy też użyć standardu ECM (*Extended Computer Modern*) zaakceptowanego podczas 5-ej Europejskiej Konferencji TeX-owej (Cork, Irlandia, 3–15 września 1990), gdyż prace nad tym standardem wciąż pozostają niezakończone. Przykładowo, nie został opublikowany zestaw niezbędnych podcięć i ligatur, a kształty znaków są niedopracowane. Ponadto TeX-owy format `plain` nie nadaje się do pracy z fontami ECM. Być może zabrakło nam odwagi, a na pewno motywacji, by pisać od początku „plainopodobny" format dla fontów ECM.

Mniejszym złem wydawało się nam rozszerzenie tabeli znaków fontów CM o polskie znaki diakrytyczne. Przedrostek CM w nazwach fontów zastąpiliśmy przedrostkiem PL, gdyż zgodnie z decyzją Knutha przedrostek CM jest zastrzeżony dla fontów oryginalnych.

Fonty tekstowe rodziny PL zawierają osiemnaście polskich liter: 'ą', 'ć', 'ę', 'ł', 'ń', 'ó', 'ś', 'ź', 'ż', 'Ą', 'Ć', 'Ę', 'Ł', 'Ń', 'Ó', 'Ś', 'Ź', 'Ż' oraz dodatkowe cudzysłowy: '„' (polski cudzysłów otwierający), '«' (francuski cudzysłów otwierający), i '»' (francuski cudzysłów zamykający). Jednakże nie dołączyliśmy tych znaków do fontu `PLTEX10`. Jego odpowiednik, font `CMTEX10` odzwierciedla układ klawiatury, na której pracował Knuth (p. [Knuth86b], str. 568). Wydaje się zupełnie nieprawdopodobne, że klawiatura ta zawierała polskie znaki. Tym samym fonty `CMTEX10` i `PLTEX10` są identyczne.

Układ dolnej połówki fontów PL (znaki o kodach <128) jest identyczny z układem fontów CM. Zachowane zostały nawet takie anomalie jak rozbieżność między układem `CMR10` a `CMR5`. Polskim znakom diakrytycznym przypisane zostały kody zgodne ze standardem ECM, zaś cudzysłowom — dość dowolnie wybrane kody >127 (mamy nadzieję, że nie okaże się któregoś dnia, że kody te zostały wybrane błędnie).

Fonty matematyczne PL nie zawierają polskich znaków diakrytycznych. Każdy z pewnością się zgodzi, że formuła „$ą^2 = ę^3$" wygląda nieco zabawnie. Jedynie font `PLSY` został poszerzony o znaki '$\leqslant$' i '$\geqslant$' używane w polskiej typografii zamiast '$\leq$' i '$\geq$'.

2.1 Zgodność z fontami CM

Mimo nieuniknionych różnic między fontami PL i CM, dążyliśmy do uzyskania możliwie najwyższej zgodności między obiema rodzinami, tzn. staraliśmy się by każdy font PL zawierał odpowiadający mu font CM.

Wiele razy walczyliśmy z pokusą zmieniania oryginalnych fontów. Na przykład chętnie zwiększylibyśmy kropki nad literami 'i' oraz 'j', zwłaszcza w fontach italikowych i imitujących maszynę do pisania fontach `CMTT`. Nie zrobiliśmy tego, respektując ważniejszą niż poczucie estetyki ideę zgodności. Inny przykład: w fontach CM nie jest wstawiane automatycznie podcięcie ani między 'A' i 'w', ani między 'A' i 'v', podczas gdy istnieją słowa angielskie w których takie pary liter mogą się pojawić, np. (*nomen omen*) „Awkward" lub „Average". Stosowne podcięcia zostałyby wygenerowane, gdyby w programach METAFONT-owych opisujących fonty PL zmiennej boole'owskiej *improve_kerns* nadać wartość *true*. Wartością domyślną tej zmiennej jest oczywiście *false* i nie przewiduje się, by w normalnych zastosowaniach wartość ta miała ulec zmianie.

Jedyne niezgodności, jakich nie udało się nam uniknąć, spowodowane są ograniczeniami systemu TeX/METAFONT: plik metryczny każdego z fontów (plik TFM) nie może zawierać więcej niż szesnaście różnych wysokości. Gdy program generuje większą liczbę różnych wysokości, METAFONT przed utworzeniem pliku metrycznego redukuje ich liczbę zastępując wymiary oryginalne wymiarami przybliżonymi.

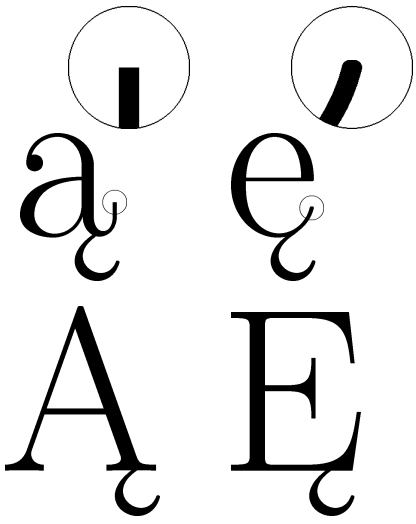
Tymczasem prawie wszystkie fonty CM zawierają już znaki mające łącznie szesnaście różnych wysokości (np. `CMR`, `CMTI`, `CMSS`, `CMCSC`, `CMTT`). Z drugiej strony wprowadzenie dodatkowych wysokości jest w przypadku polskiego języka niezbędne ze względu na majuskuły 'Ć', 'Ń', 'Ó', 'Ś', 'Ź' oraz 'Ż'. Nie wydaje się sensowne sztuczne zmniejszanie wysokości tych znaków, a dodanie choćby jednej nowej wysokości musi spowodować, że wysokości niektórych znaków w fontach PL będą się różnić od wysokości takich samych znaków w fontach CM. Oznacza to, że w pewnych — miejmy nadzieję niezwykle rzadkich — przypadkach TeX inaczej przełamie na strony tekst złożony fontami CM niż tekst złożony fontami PL. Łamanie wierszy będzie jednakże w obu wypadkach identyczne[1].

[1] Nie tylko liczba różnych wysokości w foncie jest ograniczona. Również liczba tzw. poprawek italikowych (*italic corrections*) nie może przekroczyć liczby 64. Otarliśmy się prawie o tę granicę, gdyż w fontach `PLSL10`, `PLSL12`, `PLSSI17`, `PLTI8` oraz

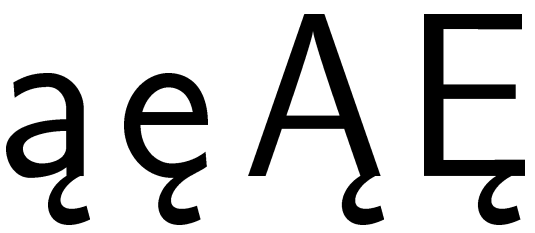
Mimo iż niezgodności nie są wielkie, czasem może się okazać, że jedynie użycie fontów CM wchodzi w rachubę. Nie oznacza to bynajmniej konieczności generowania odpowiednich fontów za pomocą METAFONT-a. Wystarczy zastosowanie fontów wirtualnych opartych o pliki metryczne fontów CM i mapy bitowe fontów PL (patrz [Knuth90b]).

2.2 Kształty znaków diakrytycznych

Najbardziej charakterystycznymi znakami dla polskiego języka są litery z ogonkiem, tzn. 'ą', 'ę', 'Ą' oraz 'Ę'. Należy podkreślić, że w istocie co najmniej trzy różne ogonki są potrzebne: inny ogonek jest potrzebny dla 'ą', inny dla 'ę', a dla 'Ą' i 'Ę' jeszcze inny. Powodem jest różny sposób połączenia litery z ogonkiem w każdym z tych przypadków[2]. Proszę także zwrócić uwagę na różne zakończenia łuków w literach 'a' oraz 'e' — sprawiło nam sporo kłopotu takie zaprojektowanie ogonka, by harmonijnie łączył się z obiema literami:

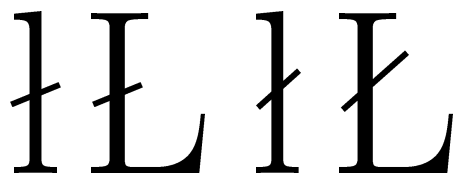


Co do rozmiarów ogonka, to zdecydowaliśmy, że duże i małe litery będą miały ogonki tej samej wielkości (z wyjątkiem fontu PLCSC). Dzięki temu w foncie bezszeryfowym wystarczają szczęśliwie dwa ogonki, gdyż ogonek przy literze 'ą' jest dokładnie taki sam jak ogonek przy 'Ą' i 'Ę':



Niektórzy typografowie wolą jednak, by rozmiary ogonków przy majuskułach były nieco większe niż przy minuskułach. Jeśli dodatkowo ogonki przy 'Ą' i 'Ę' byłyby różne (np. z powodu ekscentrycznych szeryfów — takich jak w fontach CMFF10 czy CMFI10), mielibyśmy do czynienia z czterema różnymi ogonkami. A jeśli, na przykład, chcielibyśmy uwzględnić litewskie ogonki przy 'i' i 'u', to liczba ogonków jeszcze by wzrosła.

Typowymi dla języka polskiego są także litery 'ł' i 'Ł'. Fonty CM zawierają wprawdzie znak przekreślenia dla liter 'l' i 'L', ale naszym zdaniem litery uzyskane za jego pomocą są dalekie od doskonałości. Zdecydowaliśmy się na wyraźne rozróżnienie pomiędzy przekreśleniem minuskuły i majuskuły (lewa para znaków to efekt działania makr \l i \L, prawa — to znaki z fontów PL):



Oryginalne (CM-owe) litery 'ł' i 'Ł' prowadzą z reguły do katastrofalnych efektów. Na przykład w słowie „Jagiełło" litery 'ł' są stanowczo zbyt blisko siebie:

Fonty PL zdają się okazywać nieco więcej szacunku królewskiemu nazwisku:

Akcenty nad minuskułami (litery 'ć', 'ó', 'ń', 'ś' oraz 'ź') w fontach PL są niemal identyczne

PLTI12 „wyszły" nam aż 63 różne poprawki italikowe. Gdybyśmy mieli mniej szczęścia i przekroczyli „zakazaną granicę" 64, również akapity mogłyby być łamane na wiersze inaczej w obu przypadkach.

[2] Ogonek przy 'ą' powinien w zasadzie łączyć się z literą łagodnie, podobnie jak w literze 'ę'. Jednakże w fontach CM prowadziłoby to do zbytniego przesunięcia ogonka przy 'ą' w prawo, dlatego zastosowaliśmy inne podejście.

z akcentami uzyskiwanymi za pomocą TEX-owej instrukcji \'. Natomiast akcenty nad majuskułami są nieco „spłaszczone", gdyż w przeciwnym razie litery byłyby zbyt wysokie, co utrudniałoby zachowanie odpowiednich odstępów międzywierszowych. Proszę zwrócić uwagę, że dla fontów bezszeryfowych zwykły obrót lub pochylenie (ang. *slant*) akcentu nie wystarcza — potrzebny jest nieco subtelniejszy zabieg:

ś Ś ś Ś

Pozostały nam jeszcze litery 'ż' i 'Ż'. W fontach CM kropka akcentowa uzyskiwana za pomocą instrukcji \. nie zawsze jest taka sama jak kropka nad 'i' czy 'j'. Jest to ewidentny błąd, typowy zresztą dla fontów zagranicznego pochodzenia pojawiających się na polskim rynku. Proszę porównać powiększone słowo „niż" złożone fontem CMTI (z lewej) i fontem PLTI (z prawej):

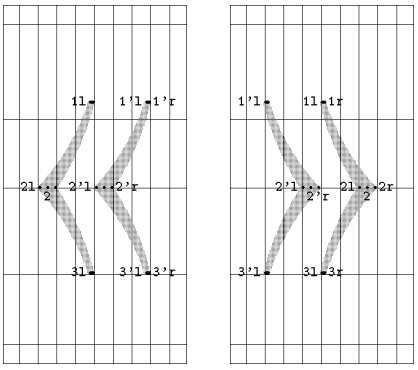
niż niż

Warto podkreślić, że w ogólnym przypadku dodanie znaków diakrytycznych może oznaczać konieczność zmiany kształtu niektórych innych znaków w foncie. Innymi słowy, zabieg typu „dorobienia akcentów" jest z reguły niewystarczający dla uzyskania fontu zawierającego znaki diakrytyczne dobrze pasujące do pozostałych znaków.

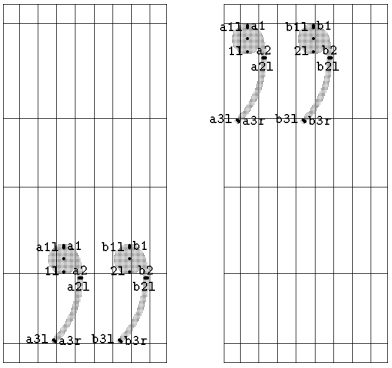
2.3 Cudzysłowy

Polski cudzysłów otwierający jest podobny do angielskiego cudzysłowu zamykającego, tyle że przesunięty w dół do podstawy znaku i umieszczony w polu znaku nieco bardziej na prawo; polski cudzysłów zamykający jest dokładnie taki sam jak cudzysłów angielski:

Cudzysłowy francuskie (używane w języku polskim wewnątrz cytatów) zostały dołożone do zestawu znaków, gdyż w fontach CM nie istnieją znaki, które do tego celu można by wykorzystać.



Nie należy mylić cudzysłowu otwierającego '«' z zamykającym '»' (por. pkt 3.2).



Nawiasem mówiąc kolejny raz mieliśmy tu szczęście. Gdyby polski cudzysłów zamykający był nie taki jak angielski lecz taki jak niemiecki, tj. '„', to byśmy się znaleźli w kłopotliwej sytuacji. Otóż CM-owa ligatura '"' nie może zostać po prostu użyta jako cudzysłów zamykający: powinna być przesunięta trochę w lewo, inne współczynniki odstępu (*space factors*) powinny zostać przypisane znakom tworzącym ligaturę (p. pkt 3.2) i tylko jeden ze znaków — albo *angielski cudzysłów otwierający* '"', albo *niemiecki zamykający* '„' — mógłby otrzymać uprzywilejowaną pozycję znaku będącego ligaturą dwóch apostrofów.

2.4 Automatycznie wstawiane podcięcia

Dodatkowe odstępy, wstawiane automatycznie przez system składu między niektórymi znakami (*implicit kerns*), są sprawą kluczową dla uzyskania druku wysokiej jakości. Nawet najpiękniej zaprojektowane litery nie pomogą, gdy odstępy te są dobrane nieprawidłowo. Z przykrością należy stwierdzić, że

podcięcia fontów CM nie są dobrane najlepiej (p. pkt 2.1). Po części wynika to ze śmiałego zamierzenia Knutha, by całą rodzinę fontów opisać jednym programem METAFONT-owym. Wydaje się nam, że jest rzeczą niemal niemożliwą odpowiednie wyważenie podcięć we wszystkich siedemdziesięciu pięciu fontach rodziny CM za pomocą wspólnej dla całej rodziny instrukcji `ligtable` (zawierającej tylko jedną instrukcję warunkową, sprawdzającą czy dany font jest fontem szeryfowym, czy też bezszeryfowym)!

Polskie znaki diakrytyczne niewiele się różnią od odpowiadających im liter łacińskich, dlatego też i odpowiednie podcięcia niewiele się różnią. Jak zawsze, tak i w tym wypadku jest trochę wyjątków. Np. bez dodatkowego odstępu między 'ą' i 'g' litery te prawie by się dotykały (sytuację bez podcięcia przedstawia para znaków z prawej strony):

ąg ąg

Wiele podcięć zostało wprowadzonych w związku z polskim cudzysłowem otwierającym '„', gdyż musi on być odsunięty od dolnych szeryfów i dosunięty do takich liter jak 'W' czy 'T'. Tym niemniej staraliśmy się zachować styl CM-owy i zawsze posłusznie przestrzegaliśmy pouczenia Knutha [Knuth86a, str. 317]:

> *„Nowicjusze często przesadzają z podcięciami. Najlepsze efekty uzyskuje się stosując podcięcia dwukrotnie mniejsze od tych, które wydają się dobre na pierwszy rzut oka, jako że podcięcia nie powinny być zauważalne (zauważalny powinien być ich brak). Podcięcia, które dobrze wyglądają w znaku firmowym lub tytule, na ogół zakłócają rytm czytania w zwykłym tekście."*

2.5 Składanie polskich tekstów z użyciem PostScript-u

Łącze TEX-PostScript jest bardzo ważnym elementem w procesie otrzymywania druków wysokiej jakości. Dlatego skupimy się przez chwilę na zagadnieniach związanych z tym łączem.

Ogólnie rzecz biorąc PostScript pozwala na użycie trzech rodzajów fontów:

- fontów reprezentowanych za pomocą map bitowych
- ładowalnych fontów obwiedniowych
- wbudowanych fontów obwiedniowych

Pierwsze dwa rodzaje fontów nie sprawiają żadnych kłopotów, tyle że — póki co — obwiedniowe PostScript-owe fonty PL nie istnieją. Na szczęście na TEX-u świat się nie kończy. Wiele firm produkuje obwiedniowe fonty PostScript-owe zawierające polskie znaki. Jest tylko jeden problem z takimi fontami, ten mianowicie, że polskie znaki diakrytyczne są na ogół niedopracowane. Jedynie fonty z firmy BitStream zdają się być wyjątkiem, przynajmniej jeśli chodzi o znaki polskie, chociaż akcenty nad majuskułami i minuskułami są identyczne (p. pkt 2.2), a podcięcia zostawiają wiele do życzenia.

Wadliwie zaprojektowane znaki narodowe stanowią wyzwanie dla lokalnych twórców fontów, skutkiem czego na rynku pojawia się mnóstwo rozmaitych odmian i przeróbek fontów oryginalnych, co z kolei rodzi kłopoty ze standaryzacją. Ale to już zupełnie inna historia.

Efekty uzyskiwane za pomocą fontów trzeciego rodzaju (wbudowanych fontów firmy Adobe) są zdecydowanie najgorsze. Jak to już zostało wyjaśnione w punkcie 2.2, są potrzebne co najmniej trzy różne ogonki, podczas gdy firma Adobe oferuje w swoich fontach wbudowanych tylko jeden (oczywiście nie ma w tych fontach gotowych polskich znaków diakrytycznych). W efekcie co najmniej jedna litera z ogonkiem musi mieć kształt „nieprofesjonalny".

Podobne zarzuty można postawić znakom akcentowanym: w fontach wbudowanych jest tylko jeden akcent *acute*, podczas gdy — jak to wyjaśnialiśmy — przydałyby się dwa. Operację „spłaszczania" akcentu można oczywiście próbować zaprogramować w języku PostScript, choć nie jest bynajmniej proste zadanie. Co więcej, PostScript (podobnie jak TEX) traktuje znaki złożone inaczej niż zwykłe litery. Na przykład wydrukowanie obwiedni znaku z doczepionym ogonkiem jest praktycznie niemożliwe, PostScript wydrukuje nałożone na siebie obwiednie litery i ogonka.

I wreszcie należy podkreślić, że *nie istnieje* jeden PostScript (inaczej niż TEX): staranne zaprogramowanie umieszczania akcentów okazać się może daremnym trudem, gdyż przy zmianie maszyny PostScript-owej ten sam program może umieścić akcenty w naprawdę zadziwiających miejscach.

Tak więc opinię Knutha [Knuth90b, str. 13]:

> *„Używając tych fontów*[3] *łatwiej składam teksty w językach europejskich za pomocą fontów Times-Roman, Helvetica czy Palatino niż za pomocą fontów Computer Modern!"*

wypada uznać za zbyt entuzjastyczną. Naszym zdaniem chcąc składać polskie teksty z wykorzystaniem PostScript-u należy unikać używania wbudowanych fontów PostScript-owych. Znacznie lepsze wyniki uzyskuje się używając fontów obwiedniowych lub bitmapowych. Dodać jeszcze tylko można, że przydałoby się łącze TEX-PostScript uzupełnić łączem METAFONT-PostScript.

Na zakończenie części dotyczącej fontów chcielibyśmy podkreślić, że gdyby nie życzliwa i wszechstronna pomoc ze strony zmarłego niedawno pana Romana Tomaszewskiego, wybitnego polskiego typografa, prezesa polskiego oddziału A-Typ-I (*Association Typographique Internationale*) nie uniknęlibyśmy mnóstwa pułapek i kto wie czy udałoby się nam doprowadzić prace do końca.

3 Ligatury uznane za niepożądane

W podstawowym podręczniku systemu TEX, zatytułowanym TEXbook, Knuth sugeruje, że mechanizm tworzenia ligatur można wykorzystać do uzyskiwania w TEX-u europejskich znaków narodowych [Knuth90a, str. 45–46]. Rozważając fonty wirtualne Knuth idzie nawet dalej [Knuth90b, str. 13–14]:

> *„Zgrabny schemat ligatur dla wielu języków europejskich został dopiero co opisany przez Haralambousa w listopadowym numerze TUGboat-a. Haralambous używa wyłącznie znaków ASCII uzyskując Aacute za pomocą kombinacji* `<A`*. Z łatwością mógłbym dodać ten schemat do mojego dopisując kilka linii w moich plikach* `vpl`*. W istocie kilka różnych konwencji można by zastosować równocześnie (choć nie polecałbym takiego podejścia)."*

Pozwalamy sobie nie zgodzić się z naszym Wielkim Szamanem: *nie polecamy żadnej konwencji ligaturowej* jako metody dostępu do znaków diakrytycznych. Pokażemy dalej, że niektóre mechanizmy TEX-a nie funkcjonują jak należy, jeśli użyć ligatur w ten właśnie sposób.

[3] Tzn. wbudowanych fontów PostScript-owych.

Weźmy dla przykładu język polski. Aby móc wprowadzać polskie znaki jako ligatury TEX powinien być przygotowany w następujący sposób:

- Należy wygenerować font zawierający wszystkie żądane znaki diakrytyczne.
- Pliki metryczne (pliki TFM) tych fontów należy tak skonstruować, by pewne ciągi znaków przekształcane były za pomocą mechanizmu ligaturowania w pojedyncze znaki, w naszym przypadku w znaki diakrytyczne; na przykład pary: `/a`, `/c`, `/e`, `/l`, `/n`, `/o`, `/s`, `/x` oraz `/z` powinny się ligaturować odpowiednio do: 'ą', 'ć', 'ę', 'ł', 'ń', 'ó', 'ś', 'ź' oraz 'ż'. Można też użyć notacji *postfiksowej*: `a/`, `c/`, `e/`, `l/`, `n/`, `o/`, `s/`, `x/`, `z/`. Analogiczne ligatury należy przygotować dla majuskuł. W grę mogą wchodzić także inne rodzaje ligatur, np. do pomyślenia jest notacja *infiksowa* w rodzaju `a/a` (co mogłoby oznaczać np. 'ą'). Haralambous we wspomnianej publikacji [Haralambous89] zaproponował ligatury potrójne. Jeszcze bardziej skomplikowany schemat został zaprojektowany przez twórców rodziny fontów cyrylicowych `WNCYR`[4].

Przyjrzyjmy się teraz konsekwencjom takiego zastosowania ligatur. Rozważymy kolejno: podcięcia, współczynniki odstępu (*space factors*) oraz parametry `\lefthyphenmin` i `\righthyphenmin`.

3.1 Ligatury kontra podcięcia

TEX automatycznie wstawia podcięcia wokół ligatur zgodnie z następującymi zasadami:

- podcięcie wstawiane między pojedynczym znakiem — powiedzmy — v, a ligaturą zestawioną ze znaków — powiedzmy — xyz jest takie samo jak podcięcie między v a x (tj. między v a pierwszym znakiem ligatury);
- podcięcie wstawiane między ligaturą xyz a pojedynczym znakiem w zależy od danej ligatury i znaku, i na ogół różni się od podcięcia między z a w.

W przypadku tradycyjnych ligatur (takich jak cudzysłowy czy ligatury zawierające literę 'f',

[4] Na przykład literę 'Щ' można uzyskać w fontach `WNCYR` na jeden z następujących sposobów: ШЧ, Шч, 6X, 6x, ШЩХ, ШЩх, ШцХ, Шцх, СХЦХ, СХЦх, СХцХ, СХцх, СхЦХ, СхЦх, СхцХ, bądź Схцх. Tajemniczą szóstkę można otrzymać jako ligaturę ШЩ bądź Шц.

tj. 'ff', 'fi', 'fl', 'ffi', 'ffl') zasady te prowadzą do pożądanych efektów, gdyż ligatura przypomina kształtem znaki, z których została zestawiona. Jeśli chodzi o znaki diakrytyczne, rzecz się ma zupełnie inaczej: kształt ligatury na ogół *nie ma wiele wspólnego* z kształtem znaków tworzących tę ligaturę.

Załóżmy dla przykładu, że polskie litery mają być reprezentowane jako ligatury zawierające znak / („ciach") oraz literę: /a, /c, /e, /l, /n, /o, /s, /x, /z. Skutek będzie taki, że podcięcie przed polską literą będzie niezależne od litery, gdyż będzie związane ze znakiem /. Gdyby zastosować jakąkolwiek inną notację *prefiksową*, np. taką w której każda litera miałaby inny prefiks, nie poprawi to w istotny sposób sytuacji.

Knuth z pewnością był świadom tego rodzaju kłopotów proponując o/ i O/ (a nie /o i /O) jako przykładowe ligatury dla norweskich znaków narodowych 'ø' i 'Ø' [Knuth90a, str. 45–46].

Niestety, notacja *postfiksowa* także nie rozwiązuje problemu.

Gdyby polskie litery były kodowane z użyciem notacji *postfiksowej*, np. jako ligatury liter i znaku / (a/, c/, e/, l/, n/, o/, s/, x/, z/), podcięcie poprzedzające ligaturę byłoby takie samo jak podcięcie związane z literą rozpoczynającą ligaturę.

Dla *niektórych* języków i dla *niektórych* fontów notacja tego typu może dać poprawne podcięcia. Niestety, w ogólnym przypadku podcięcie przed ligaturą *nie musi być* takie samo jak przed literą rozpoczynającą ligaturę. Proszę na przykład porównać słowa „kat" i „kąt" złożone tzw. fontem kancelaryjnym:

Kolizja liter 'k' i 'ą' jest nie do uniknięcia, jeśli podcięcie przed 'a' i 'ą' jest takie samo, choć w tym szczególnym przypadku zalecamy zmianę kształtu ogonka lub użycie innego wariantu litery 'k' (lub i jedno, i drugie)[5].

A zatem używanie ligatur jako notacji dla znaków diakrytycznych koliduje z podcięciami wstawianymi automatycznie przez TEX-a. Szczególnie niekorzystna jest notacja *prefiksowa*.

[5] O bardziej przekonujące przykłady łatwiej w innych językach, proszę np. pomyśleć o niemieckich słowach „Tasche" i „Täglich".

3.2 Ligatury kontra współczynniki odstępu

TEX przypisuje każdemu znakowi „współczynnik odstępu" (\sfcode), dzięki czemu odstęp po dowolnym znaku może być regulowany. Patrząc pod tym kątem można wyróżnić cztery kategorie znaków:

- Wszystkie minuskuły oraz większość pozostałych znaków mają współczynnik odstępu równy 1000, co powoduje, że jeśli słowo jest zakończone takim znakiem, to odstęp po tym znaku pozostaje bez zmian.
- Znaki przestankowe w formacie plain mają współczynnik odstępu większy niż 1000, a to oznacza, że odstęp po takim znaku ma być nieco większy od normalnego odstępu.
- Majuskuły mają współczynnik odstępu równy 999; powoduje to, że odstęp po takim znaku praktycznie się nie zmieni jeżeli między majuskułą a odstępem pojawi się znak przestankowy. Za tym nieco dziwnym prawidłem kryje się obserwacja, że jeśli po dużej literze pojawia się kropka, to najprawdopodobniej jest to koniec skrótu (np. imienia) a nie koniec zdania.
- Są również znaki o współczynniku odstępu równym 0; taki współczynnik odstępu powoduje, że znak jest „przezroczysty" dla algorytmu wyznaczającego wielkość odstępu, tzn. wielkość odstępu zależy od pierwszego licząc od końca słowa znaku o niezerowym współczynniku odstępu. Format plain współczynnik równy 0 przypisuje nawiasom zamykającym oraz apostrofowi (a tym samym i ligaturze powstającej z dwóch apostrofów).

Co to ma wspólnego z wersjami narodowymi TEX-a i z dostępem do znaków diakrytycznych *via* ligatury? Z tego co zostało powiedziane powyżej wynika, że współczynnik odstępu wszystkich znaków narodowych powinien być równy 1000 dla małych liter, a 999 dla dużych.

TEX oblicza współczynnik odstępu dla ligatury na podstawie współczynników znaków tworzących ligaturę. W przypadku notacji *prefiksowej* (/a, /c, /e, /l, /n, /o, /s, /x, /z, /A, /C, /E, /L, /N, /O, /S, /X, /Z) wszystko jest w porządku. Dla notacji *postfiksowej* należałoby użyć innego znaku dla ligatur małych liter, a innego dla dużych, np.: a/, c/, e/, l/, n/, o/, s/, x/, z/, A‘, C‘, E‘, L‘, N‘, O‘, S‘, X‘, Z‘.

Ale wówczas trzeba by nadać jednemu ze znaków (w tym wypadku „lewemu" apostrofowi) nienaturalny i niezgodny z formatem plain współczynnik odstępu równy 0 lub 999. W sposób oczywisty te znaki, które w formacie plain mają współczynnik odstępu równy 0, nie nadają się jako składniki ligatur (p. wyżej).

Jeśli chodzi o sugestię Knutha, by norweskie znaki 'ø' i 'Ø' zdefiniować jako ligatury o/ i O/, należy stwierdzić, że zawsze pojawią się kłopoty związane ze współczynnikami odstępu: albo współczynnik odstępu litery 'Ø' będzie równy 1000 zamiast 999, albo współczynnik odstępu znaku / będzie równy 0 lub 999 zamiast 1000.

Wróćmy jeszcze na moment do wspomnianej rodziny fontów WNCYR. Znak 'Ш' można otrzymać jako jedną z dwóch ligatur: SH lub Sh. Każda z nich prowadzi oczywiście do innego współczynnika odstępu. Ten sam współczynnik odstępu (999) otrzymało by się zastępując drugą z ligatur przez sH.

A zatem używanie ligatur jako notacji dla znaków diakrytycznych prowadzi na ogół do nieprawidłowych odstępów między słowami. Szczególnie niekorzystna jest notacja *postfiksowa*.

3.3 Ligatury kontra parametry \lefthyphenmin i \righthyphenmin

Ligaturę tworzą co najmniej dwa znaki. Okazuje się, że ma to nieoczekiwane i na ogół niepożądane konsekwencje jeśli chodzi o współdziałanie z parametrami \lefthyphenmin oraz \righthyphenmin.

Parametry te (dokładniej rejestry licznikowe, *count registers*) zostały wprowadzone do TEX-a 3.0. Bodźcem niewątpliwie był tu powstały wcześniej wielojęzyczny TEX M. Fergusona, w którym podobne parametry zostały wykorzystane. Załóżmy, że rejestry te zawierają liczby l (\lefthyphenmin) oraz r (\righthyphenmin). Dla algorytmu dzielenia wyrazów oznacza to, że jedynie pozycje między znakami $a_l a_{l+1} \ldots a_{n-r+1}$ słowa $a_1 a_2 \ldots a_n$ będą brane pod uwagę jako ewentualne miejsca podziału. W szczególności algorytm nie będzie próbował dzielić słów o długości $< l + r$.

Załóżmy, że słów pięcioliterowych i krótszych nie chcemy dzielić i dlatego przyjmujemy $l = r = 3$. Gdyby słowo „żądać" zapisać za pomocą ligatur jako /z/ada/c, TEX mimo wszystko mógłby podzielić to słowo po pierwszej sylabie („żą-dać"). Powodem jest sposób interpretowania słów przez algorytm przenoszenia. Otóż algorytm ten dzieli wyrazy *przed* utworzeniem ligatur. Z punktu widzenia algorytmu przenoszenia należy podzielić *ośmioznakowe* słowo /z/ada/c, i może ono zostać podzielone po *czwartym* znaku, tj. po fragmencie /z/a, gdyż ustawienie parametrów na taki podział zezwala.

A zatem używanie ligatur jako sposobu dostępu do znaków diakrytycznych prowadzi do nieprawidłowego i niejednorodnego traktowania długości słów przez algorytm przenoszenia. Parametry \lefthyphenmin i \righthyphenmin tracą swoje naturalne znaczenie. Oba rodzaje notacji, tj. *prefiksowa* i *postfiksowa* z tego punktu widzenia są nieprzydatne.

3.4 I co począć z ligaturami?

Argumenty wysunięte w punktach 3.1–3.3 wystarczyły nam, by zrezygnować z reprezentowania polskich znaków diakrytycznych za pomocą ligatur, choć listę zarzutów przeciw ligaturom można by jeszcze wydłużyć.

Na przykład wzorce dzielenia wyrazów (*hyphenation patterns*) w istotny sposób zależą od użytej notacji. Powoduje to sporo kłopotów, zwłaszcza w kontekście automatycznego przekształcania wzorców przy zmianie notacji.

Morałem z tej historii może być parafraza sformułowania Knutha [Knuth91]: *ligatury jako sposób dostępu do znaków narodowych są dopuszczalne przy następujących zastrzeżeniach: (1) albo użytkownik jest D. E. Knuthem, (2) albo nie czyni z tych ligatur żadnego użytku.*

Rzecz jasna nic nie stoi na przeszkodzie, by używać ligatur w ich naturalnym znaczeniu, tzn. do uzyskiwania pauz (myślników), cudzysłowów, ligatur zawierających literę 'f', itp.

4 Podsumowanie

Tak więc zdecydowaliśmy się na własne, inne od istniejących, podejście do narodowej adaptacji TEX-a. Problemy na jakie w trakcie pracy natrafiliśmy skutecznie nas zniechęciły do myślenia o uniwersalnej adaptacji wielojęzycznej.

Jest rzeczą znaną, że TEX został początkowo zaprojektowany jako system do pracy w jednym języku, przede wszystkim angielskim. Wersja 3.0 stanowi duży krok w stronę wielojęzyczności. Tym niemniej oryginalny TEX nie ma możliwości składania tekstów pionowo czy też od prawej do lewej; znaki akcentowane mają inny status niż znaki pojedyncze; instrukcja \hyphenation nie jest zbyt użyteczna

dla języków fleksyjnych; sztywna granica 255 dla liczby tzw. operacji (*ops*)[6] jest niebezpiecznie niska; o ograniczeniach związanych z liczbą parametrów charakteryzujących font już mówiliśmy...

Argumenty można by mnożyć, ale nie w tym rzecz. Jest oczywiste, że pojęcie języka po prostu nie zostało wbudowane w system. Nieco myląca nazwa \language została wybrana dla instrukcji zmieniającej jedynie bieżący zestaw wzorców dzielenia wyrazów. Peter Breitenlohner zwraca uwagę, że języki austriacki i niemiecki mogą korzystać z tych samych wzorców, chociaż są między nimi istotne różnice [Breitenlohner90, str. 10]. Z drugiej strony jeśli zachodzi potrzeba użycia kilku fontów o różnym rozmieszczeniu znaków narodowych, pojawia się odwrotna sytuacja: kilka różnych zestawów wzorców TEX musi przechowywać w pamięci dla jednego języka (właśnie z taką sytuacją przyszło nam się zetknąć przy tworzeniu polskiej adaptacji TEX-a).

Podzielamy opinię F. Mittelbacha, że możliwe jest *ukierunkowanie przyszłych prac w taki sposób, by kiedyś się nie okazało, że wokoło jest mnóstwo różnych, niezgodnych ze sobą systemów „opartych na TEX-u"* [Mittelbach90, str. 337]. Listę rozszerzeń zaproponowanych przez Mittelbacha poszerzylibyśmy tylko o jeden punkt — o wielojęzyczność. W przeciwnym wypadku pojawienie się, i to raczej wcześniej niż później, owych wielojęzycznych „plus-minus-TEX-ów" wydaje się nieuniknione.

Na koniec chcielibyśmy wyrazić nasze ogromne uznanie dla niezwykłej pracy wykonanej przez Knutha. Wydawać by się mogło, że stworzenie tak niezawodnego i uniwersalnego systemu jest wręcz niemożliwe. A jednak TEX istnieje i pomimo pewnych jego cech faworyzujących język angielski, dał się szczęśliwie przystosować do języka polskiego.

References

[Breitenlohner90] Peter Breitenlohner, "Using TEX 3 in a multilingual environement", *TUGboat*, Vol. 11, No. 1, str. 9–12, 1990.

[Haralambous89] Yannis Haralambous, "TEX and latin alphabet languages", *TUGboat*, Vol. 10, No. 3, str. 342–345, 1990.

[Jackowski et al. 88] Bogusław Jackowski, Tomasz Hołdys, Marek Ryćko, "With TEX to the Poles", *Proceedings of the 3rd European TEX Conference*, Exeter 1988.

[Jackowski et al. 89] Bogusław Jackowski, Marek Ryćko, "Polish TEX is ready for use", *Proceedings of the 4th European TEX Conference*, Karlsruhe 1989, (nie wydane drukiem).

[Knuth86a] Donald E. Knuth, "The METAFONTbook", *Computers & Typesetting*, Vol. C, Addison Wesley, 1986.

[Knuth86b] Donald E. Knuth, "Computer Modern Typefaces", *Computers & Typesetting*, Vol. E, Addison Wesley, 1986.

[Knuth90a] Donald E. Knuth, "The TEXbook", *Computers & Typesetting*, Vol. A, Addison Wesley, 1990, eighteenth printing.

[Knuth90b] Donald E. Knuth, "Virtual Fonts: More Fun for Grand Wizards", *TUGboat*, Vol. 11, No. 1, str. 13–23 1990.

[Knuth91] Donald E. Knuth, "TEX.WEB file, ver. 3.14", marzec 1991.

[Mittelbach90] Frank Mittelbach, "E-TEX: Guidelines for future TEX", *TUGboat*, Vol. 11, No. 3, str. 337-345, 1990.

[6] Jest to pojęcie związane ze sposobem zapamiętywania wzorców dzielenia wyrazów dla danego języka; informację w rodzaju

```
Hyphenation trie of length 11344 has
375 ops out of 750
  194 for language 1
  181 for language 0
```

TEX podaje w trakcie tworzenia plików FMT.

Od przyjaciół

Syntactic Sugar

Kees van der Laan*

Abstract

A plea is made for being honest with TEX and not imposing alien structures upon it, otherwise than via compatible extensions, or via (non-TEX) user interfaces to suit the publisher, the author, or the typist. This will facilitate the process to get (complex) publications out effectively, and typographically of high-quality.

Keywords (nested) loop, switch, array addressing, plain TEX, macro writing, education.

Introduction

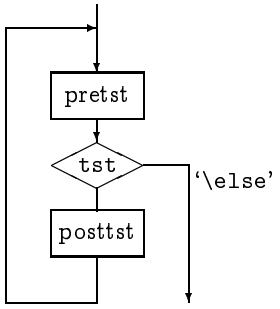
TEX is a formatter and also a programming language. TEX is different from current high-level programming languages, but very powerful. A class on its own, and therefore unusual, and unfamiliar.
Because of TEX being different, macro writers propose to harnass TEX into a more familiar system, by imposing syntaxes borrowed from various successful high-level programming languages. In doing so, injustice to TEX's nature might result, and users might become intimidated, because of the difficult—at least unusual—encoding used to achieve the aim. The more so when functional equivalents are already there, although perhaps hidden, and not tagged by familiar names. This is demonstrated with examples about the loop, the switch, array addressing, optional and keyword parameters.
Furthermore, TEX encodings are sometimes peculiar, different from the familiar algorithms, possibly because macro writers are captivated by the mouth processing capabilities of TEX. Users who don't care so much about TEX's programming power, but who are attracted by the typesetting quality, which can be obtained with TEX as formatter, can be led astray when in search for a particular functionality they stumble upon unusual encodings. They might conclude that TEX is too difficult, too error-prone and more things like that and flee towards Word*whatever*, or embrace Desk Top Publishing systems.

* This is an abridged version of the paper submitted to EuroTEX'93

The way out is education, next to the provision of compatible, well-documented and supported user interfaces, which don't act like syntactic sugar, by neglecting, or hiding, the already available functional equivalents. Neither the publication of encodings, nor the provision of encodings via file servers or archives—although a nice supporting feature for the TEX-ies—is enough. The quality, compatibility and the simplicity of the (generic) macros should be warranted too.
It is not the aim of this paper to revitalize a programming languages notation war, but to stimulate awareness, and exchange ideas.

Loops

DEK's loop, TEXbook p.219, implements the flow



with syntax

\loop⟨pretst⟩\iftst⟨posttst⟩\repeat.

Special cases result when ⟨pretst⟩, or ⟨posttst⟩ is empty. The former is equivalent to for example PASCAL's **while** ... **do** ..., and the latter to **repeat**... **until**. With this awareness, I consider the variants as proposed by for example Pittman, [22], and Spivak, [26], as syntactic sugar.
If \ifcase... is used, then we have for ⟨posttst⟩ several parallel paths, of which one—determined dynamically—will be traversed. Provide and choose your path! What do you mean by traversing the \else-path?

Why another loop?

Kabelschacht, [10], and Spivak, [26], favour a loop which allows the use of \else.[1] I have some objections to Kabelschacht's claim that his loop is a *generalization* of plain's loop.
First, it is not a generalization, just a clever, but variant, implementation of the loop flow chart.

[1] Their loops are equivalent to the general form of the loop with the execution of an extra part after the loop.

Second, it is not compatible with plain's loop. His exit path is via the \then branch (or via any of the \or-s, when \ifcase is used), and not via the \else branch.

The reason, I can think of, for introducing another loop, while the most general form has been implemented already, is the existence of commands like \ifvoid, and \ifeof, and the absence of their negatives \ifnvoid, respectively \ifneof. In those cases we like to continue the loop via the \else branch. For the latter case this means to continue the loop when the file is *not* ended. This can be attained via modifying the loop, of course, but I consider it simpler to use a \newif parameter, better known as 'boolean' or 'logical' in other programming languages. With the \newif parameter, \ifneof, the loop test for an end of file—functionally ¬\ifeof—can be obtained via

```
\ifeof\neoffalse\else\neoftrue\fi\ifneof
```

For an example of use, see the Sort It Out subsection. Related to the above encoding of the logical ¬, are the encodings of the logical and, ∧, and or, ∨, via

Functional code	TEX encoding
¬\if...	\if...\notfalse\else \nottrue\fi\ifnot
\if...∧\if...	\andtrue\if...\if... \else\andfalse \else\andfalse\fi\fi \ifand
\if...∨\if...	\ortrue \if...\else\if...\else \orfalse\fi\fi \ifor

with the \newif-s: \ifnot, \ifand, and \ifor.

Nesting of loops

Pittman, [22], argued that there is a need for other loop encodings.

> 'Recently, I encountered an application that required a set of nested loops and local-only assignments and definitions.
> TEX's \loop...\repeat construction proved to be inadequate because of the requirement that the inner loop be grouped.'

If we take his (multiplication) table—I like to classify these as deterministic tables, because the data as such are not typed in—to be representative, then below a variant encoding is given, which does not need Pittman's double looping. The table is typographically a trifle, but it is all about how the deterministic data are encoded. My approach is to consider it primarily as a table, which it is after all. Within the table the rows and columns are generated, via recursion, and not via the \loop. Furthermore, I prefer to treat rules, a frame, a header and row stubs as separate items to be added to the table proper, [15]. The *creation* of local quantities is a general TEX aspect. I too like the idea of a hidden counter, and the next best TEX solution via the local counter. The local versus global creation of counters is a matter of taste, although very convenient now and then. The creation of local quantities is tacitly discouraged by DEK's implementation, because there is no explicit garbage collector implemented and therefore no memory savings can be gained. The only thing that remains is protection against programming mistakes, which is indeed important.

Pittman's table, focused at the essential issue of generating the elements, can be obtained via

```
$$\vbox{\halign{&\ \hfil#\hfil\strut\cr
\rows}}$$
```

with

```
\newcount\rcnt\newcount\ccnt\newcount\tnum
\newcount\mrow\newcount\mcol \mrow2 \mcol3
\def\rows{\global\advance\rcnt1
  \global\ccnt0\cols\ifnum\rcnt=\mrow\swor
  \fi\rs\rows}
\def\swor#1\rows{\fi\crcr}
\def\cols{\global\advance\ccnt1
  \tnum\rcnt\multiply\tnum\ccnt\the\tnum
  \ifnum\ccnt=\mcol\sloc\fi\cs\cols}
\def\sloc#1\cols{\fi}
\def\rs{\cr}\def\cs{&}
```

The result is

1 2 3
2 4 6

The termination of the recursion is unusual. It is similar to the mechanism used on page 379 of the TEXbook, in the macro \deleterightmost. The latter TEXnique is elaborated in [4], and [16].

The above shows how to generate in TEX deterministic tables, where the table entries in other programming languages are generally generated via nested loops. One can apply this to other deterministic math tables—trigonometric tables for example—but then we need more advanced arithmetic facilities in TEX (or inputting the data calculated by

other tools), not to mention the appropriate mapping of tables which extend the page boundaries.

For a more complete encoding see Table Diversions, [15]. The idea is that rules and a frame be commanded via \ruled, and \framed. The header via an appropriate definition of \header, ×, the indication that we deal with a multiplication table, in \first, and the row stubs via definition of the row stub list. All independent and separate from the table proper part.

A better example of a nested loop is for example the encoding of bubble sort as given in [17].

Loops and novices

Novice TEXies find DEK's loop unusual, so they sugar it into the more familiar **while**, **repeat**, or **for** constructs, encouraged to do so by exercises as part of courseware. From the functionality viewpoint, there is no need for another loop notation.

With respect to the **for** loop, I personally like the idea of a hidden counter, [13], and [22]. The hidden counter has been used in an *additional* way to plain's loop in for example [13] (via \preloop and \postloop), and will not be repeated here. This way of doing is a matter of taste, which does not harm, nor hinder, because it is a compatible extension.

And, ...for the nesting of loops we need scope braces, because of the parameter separator \repeat. If braces are omitted, the first \repeat is mistaken for the outer one, with the result that the text of the outer loop will *not* become the first \body. The good way is, to make the inner \repeat invisible at the first loop level, by enclosing the inner loop in braces.

With non-explicit nesting—for example the inner loop is the replacement text of a macro—we still need scope braces, because otherwise the \body of the outer loop will be silently redefined by the body of the inner loop.

The point I like to get accross is, that there is no real need for another loop encoding. Syntactic sugar? Yes!

Switches, is there a need?

Apart from the \ifcase... construct, TEX seems to lack a multiple branching facility with symbolic names. Fine, [4], introduced therefore

```
\def\fruit#1{\switch\if#1\is
a \apple
b \banana
c \cherry
d \date       \end}
```

I have 2, or rather 3, remarks to the above.

First, the 'switch'-functionality is already there.

Second, Fine's implementation is based upon

> 'It is clear that \switch must go through the alternatives one after another, reproducing the test...'

Well, ...going through the alternatives one after another is not necessary.

Third, his example, borrowed from Schwarz, [24], can be solved more elegantly without using a 'switch' or nested \if-s at all, as shown below.

The first two aspects are related. Fine's functionality can be obtained via

```
\def\fruit#1{\csname fruit#1\endcsname}
%with
\def\fruita{\apple}
\def\fruitb{\banana} %et cetera
```

With for example:
\def\apple{{\bf apple}},
\fruit a
yields
apple.

And what about the 'else' part? Thanks to \csname, \relax will return when the control sequence has not yet been defined. So, if nothing has to happen we are fine. In the other situations one could define \def\fruitelse{...}, and make the else fruits refer to it, for example \def\fruity{\fruitelse}, \def\fruitz{\fruitelse}, etc. When the set is really uncountable we are in trouble, but I don't know of such situations. And, ...the five letters 'fruit' are there only to enhance uniqueness of the names.

As example J. Fine gives the problem, treated by Schwarz, [24], to print vowels in bold face.[2]

The problem can be split into two parts. First, the general part of going character by character through a string, and second, to decide whether the character at hand is a vowel or not.

For the first part use for example, \dolist, TEXbook Exercise 11.5, or \fifo, [16].

```
\def\fifo#1{\ifx\ofif#1\ofif\fi#1\fifo}
\def\ofif#1\fifo{\fi}
```

[2] A bit misplaced example because the actions in the branches don't differ, except for the non-vowel part.

For the second part, combine the vowels into a string, aeiou, and the problem is reduced to the question ⟨char⟩ ∈ aeiou? Earlier, I used the latter approach when searching for a card in a bridge hand, [12].[3] That was well-hidden under several piles of cards, I presume? Recently, I have used the same method for recognizing accents and control sequences in a word, [17]. Anyway, searching for a letter in a string can be based upon \atest, TEXbook, p.375, or one might benefit from \ismember, p.379. I composed the following

```
\newif\iffound
\def\loc#1#2{%locate #1 in #2
\def\locate##1#1##2\end{\ifx\empty##2%
\empty\foundfalse\else\foundtrue\fi}
\locate#2.#1\end}
```

Then

```
\fifo Audacious\ofif
```

yields

Audacious, with

```
\def\process#1{\uppercase{\loc#1}%
{AEIOU}\iffound{\bf#1}\else#1\fi}
\def\fifo#1{\ifx\ofif#1\ofif\fi
  \process#1\fifo}
```

Note that en-passant we also accounted for uppercase vowels. By the way, did you figure out why a period—a free symbol—was inserted between the arguments for \locate? It is not needed in this example.[4] Due to the period one can test for substrings: $string_1 \in string_2$? Because, $\{string_1 \in string_2\} \wedge \{string_2 \in string_1\} \Rightarrow \{string_1 = string_2\}$, we also have the possibility to test for equality of strings, via \loc. Happily, there exists the following straightforward, and TEX-specific, way of testing for equality of strings

```
\def\eq#1#2{\def\st{#1}\def\nd{#2}
\ifx\st\nd\eqtrue\else\eqfalse\fi}
```

For lexicographic comparison, see [17], [16].

Knuth's switches

Don Knuth needed switches is his manmac macros—\syntaxswitch,\xrefswitch etc.—TEXbook,p.424. He has implemented the functionality via nested \if-s. My approach can be used there too, but with some care with respect to { token in \next (read: some catcode adaptations). For example[5]

```
\ea\def\csname sw[\endcsname{[-branch}
\ea\def\csname sw|\endcsname{bar-branche}
%etc. then
\def\next{[}\csname sw\next\endcsname, and
\def\next{|}\csname sw\next\endcsname
```

yields: [**-branch**, and **bar-branche**.

For manmac see the TEXbook, p. 412–425, and the discussion [19].

Array addressing

Related to the switch, or the old computed goto as it was called in FORTRAN, is array addressing. In TEX this can be done via the use of \csname. An array element, for example elements identified among others in PASCAL by a[1] or a[apple], can be denoted in TEX via the control sequences

```
\csname a1\endcsname
%respectively
\csname aapple\endcsname
```

For practical purposes this accessing, or should we say 'reading,' has to be augmented with macros for writing, as given in [5], and [7]. Writing to an array element can be done via

```
\def\a#1#2{\ea\def\csname a#1%
\endcsname{#2}}\a{1}{Contents}
```

yields Contents, after the above.

The point I like to make is, that 'array addressing'—also called table lookup by some authors—is already there, although unusual and a bit hidden, but, ... we are used to things like strong type-checking, isn't? Once we can do array addressing we can encode all kind of algorithms, which make use of the array data structure. What about sorting? See the Sort It Out subsection, for a glimpse, and the in depth treatment, [17], with $O(n \log n)$ algorithms, and application to glossary and index sorting.

Conclusion

It is hoped that authors who can't resist the challenge to impose syntaxes from successful programming languages upon TEX, also encode the desired functionality in TEX's peculiar way, and contrast this with their proposed improvements. The novice, the layman and his peers will benefit from it.

The difficulties caused by TEX's unusual encoding

[3] The macro there was called \strip.

[4] If omitted the find of 'bb' in 'ab' goes wrong: abbb vs. ab.bb, will be searched.

[5] \ea – an abbreviation for \expandafter

mechanisms, can best be solved via education, and not via imposing structures from other languages. The latter will entail confusion, because of all those varieties. Furthermore, it is opposed to the Reduced Instruction Set idea, which I like. For me it is similar to the axioms-and-theorems structure in math, with a minimal number of axions, all mutual orthogonal.

Publishing houses, user groups, and macro writers are encouraged to develop and maintain 'user interfaces,' which do justice to TEX's nature, and don't increase the complexity of TEX's components. Good examples are: TUGboat's sty files, AMS-LATEX & $\mathcal{AMS}$-TEX, and L$^{A}\mathcal{MS}$-TEX, and not to forget good old manmac! Macro-TEX and lxiii[6] are promising.

File servers and archives are welcomed, but the compatibility, the simplicity and in general the quality, must be warranted too. Not to mention pleasant documentation and up-to-date-ness.

My wishful thinking is to have intelligent local archives, which have in store what is locally generally needed, and know about what is available elsewhere. The delivery should be transparent, and independent whether it comes from elsewhere or was in store.

References

[1] Beebe, N.H.F (1991): The TUGlib server. MAPS 91.2, 117–123.

[2] Beeton, B.N, R. Whitney (1989): *TUGboat* 10, no. (3), 378–385.

[3] Eijkhout, V (1991): TEX by Topic. A-W.

[4] Fine, J (1992): Some basic control macros for TEX. *TUGboat* 13, no. (1), 75–83.

[5] Greene, A. M (1989): TEXreation—Playing games with TEX's mind. TUG89. *TUGboat* 10, no. (4), 691–705.

[6] Hendrickson, A (1989): MacroTEX.

[7] Hendrickson, A (1990): Getting TEXnical: Insights into TEX macro writing techniques. *TUGboat* 11, no. (3), 359–370.

[8] Jeffreys, A (1990): Lists in TEX's mouth. *TUGboat* 11, no. (2), 237–244.

[9] Jensen, K, N. Wirth (1975): PASCAL user manual and report. Springer-Verlag. *TUGboat* 11, no. (2), 237–244.

[10] Kabelschacht, A (1987): `\expandafter` vs. `\let` and `\def` in conditionals and a generalization of plain's `\loop`. *TUGboat* 8, no. (2), 184–185.

[11] Knuth, D.E (1984): *The TEXbook*, A-W.

[12] Laan, C.G van der (1990): Typesetting Bridge via TEX. *TUGboat* 11, no. (2), 265–276.

[13] Laan, C.G van der (1992a): Tower of Hanoi, revisited. *TUGboat* 13, no. (1), 91–94.

[14] Laan, C.G van der (1992b): FIFO & LIFO incognito. EuroTEX '92, 225–234. Also MAPS92.1. An elaborated version is FIFO & LIFO sing the BLUes.

[15] Laan, C.G van der (1992c): Table Diversions. EuroTEX '92, 191–211. A little adapted in MAPS92.2.

[16] Laan, C.G van der (1992d): FIFO & LIFO sing the BLUes. MAPS92.2, 139–144. (To appear TUGboat, 14, 1.)

[17] Laan, C.G van der (1993a): Sorting in BLUe. MAPS93.1. (21p. Heap sort encoding is released in MAPS92.2.)

[18] Laan, C.G van der (1993b): Typesetting number sequences. MAPS93.1. (4p.)

[19] Laan, C.G van der (1993c): Manmac BLUes. MAPS93.1. (In progress in the same series: AMS BLUes, and TUGboat BLUes.)

[20] Lamport, L (1986): LATEX, user's guide & reference manual. A-W.

[21] Maus, S (1991): An expansion power lemma. *TUGboat* 12, no. (2), 277.

[22] Pittman, J.E (1988): Loopy.TEX. *TUGboat* 9, no. (3), 289–291.

[23] Salomon, D (1992): NTG's Advanced TEX course: Insights & Hindsights. MAPS 92 Special. 254p.

[24] Schwarz, N (1987): *Einführung in TEX*, A-W.

[25] Siebenmann, L (1992): Elementary Text Processing and Parsing in TEX. *TUGboat* 13, no. (1), 62–73.

[26] Spivak, M.D (1987): L$\mathcal{AMS}$-TEX. TEXplorators.

[27] Youngen, R.E (1992): TEX-based production at AMS. MAPS92.2. 7p.

Kees van der Laan jest prezesem prężnej, holenderskiej grupy użytkowników TEX-a (NTG), z którą GUST utrzymuje ścisły kontakt.

[6] The LATEX3 project.

Polska Grupa Użytkowników Systemu TEX

adres do korespondencji:

Instytut Badań Systemowych PAN

ul. Newelska 6, 01-447 Warszawa

e-mail:

gust@camk.edu.pl (Internet)
gust@plcamk61 (Bitnet)

Polska Grupa Użytkowników Systemu TEX „GUST" została powołana przez 39 osób obecnych 8 stycznia 1992 roku na zebraniu założycielskim. Grupa została formalnie zarejestrowana przez Sąd Wojewódzki w Warszawie w dniu 8 maja 1992 roku.

Cele Grupy są określone w II rozdziale Statutu:

Art. 7. *Celem Grupy jest:*
1) *zrzeszanie użytkowników systemu TEX;*
2) *upowszechnianie systemu TEX, systemu generowania fontów* METAFONT*, ich środowiska, jak również związanego z nimi oprogramowania stanowiącego tzw. dobro wspólne (public domain);*
3) *propagowanie ochrony praw autorskich;*
4) *reprezentowanie członków Grupy, ich opinii i potrzeb;*
5) *ułatwianie kontaktów między członkami Grupy.*

Art. 8. *Grupa osiąga swoje cele przez:*
1) *współdziałanie z instytucjami, towarzystwami naukowymi oraz stowarzyszeniami, tak krajowymi jak i zagranicznymi;*
2) *inspirowanie, wspieranie i patronowanie działalności mającej na celu upowszechnianie systemów TEX i* METAFONT *oraz rozwój ich środowiska;*
3) *prowadzenie działalności szkoleniowej poprzez organizowanie kursów, konferencji, odczytów, wystaw i pokazów w dziedzinach objętych działalnością Grupy;*
4) *prowadzenie działalności wydawniczej;*
5) *ułatwianie wymiany informacji poprzez tworzenie archiwów i prowadzenie dystrybucji oprogramowania public domain związanego z systemami TEX i* METAFONT.
6) *rozwijanie innych form działalności merytorycznej służącej realizacji celów statutowych.*

I Walne Zebranie GUST-u odbyło się w dniu 5 czerwca 1992 roku w Warszawie i wzięło w nim udział ponad 60 osób. Podczas Zebrania zostały wybrane władze Grupy, w tym zarząd w następującym składzie:

- Hanna Kołodziejska (Warszawa) — prezes
- Włodzimierz Bzyl (Gdańsk)
- Marek Kowalówka (Katowice)
- Krzysztof Leszczyński (Warszawa)
- Jerzy Ludwichowski (Toruń)
- Włodzimierz J. Martin (Gdańsk)
- Stefan Paszkowski (Wrocław)
- Marek Ryćko (Warszawa)
- Janusz Sosnowski (Warszawa)

Wysokość składek została ustalona na I Walnym Zebraniu: wpisowe (jednorazowe) 100.000,- zł, składka roczna 100.000,- zł. W 1992 roku członkowie będą płacili wpisowe + połowę składki rocznej, czyli w sumie 150.000,- zł. Studentom przysługuje zniżka 50% zarówno od wpisowego, jak i składki rocznej. Składki będą przyjmowane z chwilą założenia konta w banku. Zgodnie z ustaleniami przyjętymi na Zebraniu, składki należy opłacić w ciągu miesiąca od otrzymania od nas zawiadomienia. Składka członkowska nie obejmuje prenumeraty biuletynu, obowiązkowej dla członków GUST-u.

W biuletynie GUST-u będziemy publikować artykuły na temat TEX-a, informacje o dostępnym oprogramowaniu TEX-owym, a także aktualną listę członków. Adres redakcji:

Włodzimierz Bzyl
Redakcja biuletynu GUST-u
Instytut Matematyki Uniwersytetu Gdańskiego
ul. Wita Stwosza 57, 80-952 Gdańsk
tel.: 41-52-41 w.163
e-mail: matwb@halina.univ.gda.pl

Wszystkich użytkowników TEX-a serdecznie zapraszamy do naszego stowarzyszenia!

W imieniu zarządu GUST-u: Hanna Kołodziejska (prezes)