Numero 30
Ottobre
2020

ArsTEXnica

Rivista italiana di TEX e LATEX

GUIT

http://www.guitex.org/arstexnica/

TEXnica Ars

GUIT – Gruppo Utilizzatori Italiani di TEX

ArsTEXnica è la pubblicazione ufficiale del GUIT

ArsTEXnica è la prima rivista italiana dedicata a TEX, a LATEX ed alla tipografia digitale. Lo scopo che la rivista si prefigge è quello di diventare uno dei principali canali italiani di diffusione di informazioni e conoscenze sul programma ideato quasi trent'anni fa da Donald Knuth.

Le uscite avranno, almeno inizialmente, cadenza semestrale e verranno pubblicate nei mesi di Aprile e Ottobre. In particolare, la seconda uscita dell'anno conterrà gli Atti del Convegno Annuale del GUIT, che si tiene in quel periodo.

La rivista è aperta al contributo di tutti coloro che vogliano partecipare con un proprio articolo. Questo dovrà essere inviato alla redazione di ArsTEXnica, per essere sottoposto alla valutazione di recensori. È necessario che gli autori utilizzino la classe di documento ufficiale della rivista; l'autore troverà raccomandazioni e istruzioni più dettagliate all'interno del file di esempio (`.tex`). Tutto il materiale è reperibile all'indirizzo web della rivista.

Gli articoli potranno trattare di qualsiasi argomento inerente al mondo di TEX e LATEX e non dovranno necessariamente essere indirizzati ad un pubblico esperto. In particolare tutorials, rassegne e analisi comparate di pacchetti di uso comune, studi di applicazioni reali, saranno bene accetti, così come articoli riguardanti l'interazione con altre tecnologie correlate.

Di volta in volta verrà fissato, e reso pubblico sulla pagina web, un termine di scadenza per la presentazione degli articoli da pubblicare nel numero in preparazione della rivista. Tuttavia gli articoli potranno essere inviati in qualsiasi momento e troveranno collocazione, eventualmente, nei numeri seguenti.

Chiunque, poi, volesse collaborare con la rivista a qualsiasi titolo (recensore, revisore di bozze, grafico, etc.) può contattare la redazione all'indirizzo:

arstexnica@guitex.org.

Nota sul Copyright

Associarsi a GUIT

Fornire il tuo contributo a quest'iniziativa come membro, e non solo come semplice utente, è un presupposto fondamentale per aiutare la diffusione di TEX e LATEX anche nel nostro paese. L'adesione al Gruppo prevede una quota di iscrizione annuale diversificata: 30,00 € soci ordinari, 20,00 (12,00) € studenti (junior), 75,00 € Enti e Istituzioni.

Indirizzi

Gruppo Utilizzatori Italiani di TEX
c/o Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Industriale
Via Claudio 21
80125 Napoli – Italia
`http://www.guitex.org`
guit@guitex.org

Redazione ArsTEXnica:
`http://www.guitex.org/arstexnica/`
arstexnica@guitex.org

ISSN 1828-2350 (Stampa)
ISSN 1828-2369 (Online)

15 Ottobre 2020

In quest caso, aver udit né tra' m storici, n esecrabile lamente f le fu cos pur mi ve pensarlo, con quale io o possa re la turp

GUIT meeting 2020

DICIASSETTESIMO CONVEGNO NAZIONALE SU TEX, LATEX E TIPOGRAFIA DIGITALE

31 ottobre 2020

SISSA medialab,

online

Programma del convegno

9:30 Messaggio di benvenuto. Inizio dei lavori.

9:40 *`Graphpaper`: una classe per carte da grafici*. Claudio Beccari, Francesco Biccari.
10:10 *TEX in church: more adventures*. Paulo Roberto Massa Cereda.
10:40 *Which Success for TEX as an Old Program?*. Jean Michel Hufflen.
11:10 *`europasscv`: una classe non ufficiale per il curriculum vitae nel formato Europass*. Giacomo Mazzamuto.
11:40 *Funzioni e expl3*. Enrico Gregorio.

12:10 Chiusura dei lavori. Arrivederci.
12:20 Chat.
12:50 Riunione del consiglio direttivo del Gruppo Utilizzatori Italiani di TEX.

La partecipazione è libera e gratuita, si consiglia di registrarsi entro il 30 ottobre.
Registrazione online: `www.guitex.org/home/meeting`

GUIT Gruppo Utilizzatori Italiani di TEX

`www.guitex.org`

SISSA medialab

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 30, Ottobre 2020

Gruppo Utilizzatori Italiani di TEX

Editoriale

Francesco Biccari

Tutti gli anni il numero autunnale di ArsTEXnica raccoglie gli atti del GUIT meeting, l'annuale conferenza italiana di TEX e LATEX organizzata dal GUIT e da un comitato di organizzatori dell'ente ospitante.

Quest'anno il GUIT meeting doveva tenersi il 31 ottobre a Trieste, presso la Scuola Internazionale Superiore di Studi Avanzati (SISSA), grazie al supporto del Sissa Medialab, una società della SISSA che si occupa di divulgazione scientifica. Purtroppo però l'augurio del precedente editoriale non si è concretizzato: dopo la tregua estiva dal COVID, i contagi sono ricominciati a salire vertiginosamente da settembre, obbligandoci quindi a svolgere il GUIT meeting in modalità telematica con la diretta in *streaming* su YouTube. Nonostante questo, la conferenza si è svolta senza intoppi, grazie anche all'organizzazione tecnica del Sissa Medialab, nelle persone di Matteo Gamboz, Giorgia Del Bianco, e Mick McElroy. Ci sono stati cinque interventi. Potete trovare i pdf delle presentazioni e i video della conferenza sulla pagina del GUIT meeting[1].

Vediamo quali sono quindi i cinque articoli che troverete in questo numero.

Apriamo con la presentazione di una nuova classe LATEX, Graphpaper, di Claudio Beccari e Francesco Biccari. Graphpaper permette di comporre e personalizzare diversi tipi di carta da grafici: carta millimetrata, semilogaritmica, log-log, carta polare lineare e logaritmica, carta di Smith. Tutte le carte da grafico sono composte col solo ambiente picture. Da notare che finora non era mai stato prodotto un pacchetto dedicato a questo scopo.

Paulo Roberto Massa Cereda continua la sua opera di applicazione degli strumenti del mondo TEX alla tipografia ecclesiastica: da libri di canti religiosi a libretti per la messa, dal bollettino mensile della Chiesa fino a libretti con partiture musicali, per finire con la ricomposizione dell'enciclica *Fratelli tutti* di Papa Francesco con un nuovo stile minimalista.

Il terzo articolo è di carattere più generale. L'autore Jean Michel Hufflen presenta in maniera critica alcuni aspetti della sintassi del linguaggio TEX e dei suoi derivati, facendo risalire alcune incongruenze al periodo storico in cui si è sviluppato il TEX.

Con il quarto articolo Giacomo Mazzamuto ci presenta la classe LATEX europasscv per la composizione di un curriculum vitae nel formato Europass CV, il modello standard raccomandato dalla Commissione Europea. Il formato Europass CV, lanciato nel 2002, ha visto nel 2013 un aggiornamento dello stile che lo ha reso più moderno oltre che più pulito e compatto. europasscv si riferisce proprio alla versione 2013 del formato Europass, del quale implementa soprattutto lo stile lasciando piena libertà all'utente su come strutturare i contenuti.

Nel quinto e ultimo articolo Enrico Gregorio ci farà entrare invece nel futuro di LATEX, esponendo alcune nozioni relative alle *funzioni* nel linguaggio sperimentale LATEX3 (expl3): il loro ruolo, come si definiscono, e quali sono le possibili varianti di una funzione.

Chiudo ricordando che il 29 giugno 2020 è stato il centenario della nascita del celebre disegnatore di caratteri Aldo Novarese (1920-1955).

Vi auguro una buona lettura, dandovi appuntamento al numero primaverile che porterà con sé diverse novità nella veste grafica della rivista.

▷ Francesco Biccari
Dipartimento di Fisica e Astronomia
Università degli Studi di Firenze
Firenze, Italia
biccari at gmail dot com

1. https://www.guitex.org/home/it/guit-meeting-2020

Graphpaper: una classe per carte da grafici

Claudio Beccari, Francesco Biccari

Sommario

Graphpaper è una classe LaTeX che permette di comporre e personalizzare diversi tipi di carta da grafici: carta millimetrata, semilogaritmica, log-log, carta polare lineare e logaritmica, carta di Smith. Tutte le carte da grafico sono composte col solo ambiente picture. In questo articolo verranno mostrate tutte le funzionalità di graphpaper e diversi esempi d'uso.

Abstract

Graphpaper is a LaTeX class to draw and customize several kinds of plotting paper; millimeter, semilog, log-log, linear or logarithmic polar plotting papers, and the Smith chart. All such papers are typeset with the facilities of the picture environment and its extensions. In this article we show all the features of the graphpaper class and several usage examples.

1 Introduzione

Le carte da grafico sono da sempre usate in ambito tecnico-scientifico per aiutare nella rappresentazione di valori (Wikipedia, b). Un esempio molto noto è la carta millimetrata (in realtà il nome corretto sarebbe carta quadrettata, dove l'aggettivo millimetrata si usa quando il lato del quadretto più piccolo è 1 mm), che si trova facilmente in commercio. Un altro esempio è la carta semilogaritmica in cui un asse ha una scala lineare mentre l'altro ha una scala, appunto, logaritmica. È tipicamente usata per rappresentare relazioni esponenziali ($y = Ae^{bx}$), che quindi vengono linearizzate su questo tipo di carta. Analogamente, la carta log-log ha entrambi gli assi in scala logaritmica ed è quindi utile per rappresentare relazioni di tipo potenza ($y = Ax^k$), che in questo caso vengono linearizzate. Le carte polari, sia lineari che logaritmiche, sono utili quando i dati da mettere in grafico hanno naturalmente una descrizione in coordinate polari. Infine la carta di Smith (Wikipedia, a), molto usata dagli ingegneri elettrici per le linee di trasmissione, è in realtà qualcosa di più di un semplice insieme di linee per rappresentare dei dati, ma è un vero e proprio strumento di calcolo grafico.

A parte la carta millimetrata, attualmente è praticamente impossibile trovare in commercio tutte le altre, dato che oramai i software per mettere in grafico e analizzare i dati sono diventati alla portata di tutti.

In alcuni corsi universitari di laboratorio vengono però, talvolta, ancora usate, soprattutto la carta millimetrata, la semilogaritmica, e la carta di Smith. Il motivo è principalmente di tipo didattico, perché avere un contatto fisico con il grafico permette di dare un'attenzione maggiore a molti dettagli (la scala, le unità di misura, le etichette) di cui altrimenti verrebbe trascurata l'importanza, se si lasciassero scegliere automaticamente a un software.

E qui si innesta graphpaper. Questa classe nasce da un post pubblicato sul forum del GuIt del 10 agosto 2019 (Biccari). FB aveva bisogno di carta lineare (millimetrata) e semilog per il suo corso all'università, ma non voleva usare file presi da internet, difficilmente personalizzabili e di aspetto poco professionale, e non voleva stampare con stampanti di bassa qualità, con notevole spreco di risorse. La richiesta del post era quindi di sapere se qualcuno avesse già preparato un codice o un pacchetto apposito per comporre questo tipo di carta da grafici, in modo poi da far stampare i relativi pdf in una tipografia.

La risposta fu negativa e quindi da lì è cominciata una collaborazione tra FB e CB che ha portato alla realizzazione di una classe LaTeX apposita per la composizione di carte da grafici, graphpaper. Infatti, attualmente, per realizzare delle carte da grafico non ci sono strumenti dedicati, ma solamente delle soluzioni per alcuni casi specifici. Per esempio c'è il pacchetto graphpap che però permette solamente di realizzare una griglia rettangolare, oppure pgfplots (Feuersänger, 2020) che permette di creare una bellissima carta di Smith.

Per la realizzazione di graphpaper gli autori hanno usato l'ambiente picture e le sue estensioni e il pacchetto xfp per i calcoli in virgola mobile. La classe viene presentata al pubblico per la prima volta in questo articolo ed è stata pubblicata pochi giorni fa sul CTAN (Beccari e Biccari, 2020).

2 Guida all'uso

Graphpaper è abbastanza versatile da adattarsi a diversi scenari. Inizieremo presentando il suo uso più semplice: la composizione di una o più carte da grafico, con le opzioni di default, per poi essere stampate e distribuite agli studenti. Più avanti mostreremo come personalizzare colori, dimensioni, e altri aspetti delle carte. Per finire, vedremo che è possibile usare graphpaper per disegnare ciò che si vuole sulle carte da grafico, grazie alle funzionalità dell'ambiente picture.

2.1 Uso basilare

Come ogni classe LaTeX, graphpaper si carica in questo modo:

```
\documentclass[⟨options⟩]{graphpaper}
```

Ha tre opzioni di classe, mutuamente esclusive, che servono a specificare il formato della carta: a4paper (default), a3paper, letterpaper. L'orientamento del foglio è *landscape* e non può essere cambiato.

Per disegnare le varie carte da grafico, graphpaper fornisce un comando per ogni tipo di carta. Questi comandi vanno usati all'interno del corpo del documento. Alla fine di ciascuno di questi viene automaticamente inserita un'interruzione di pagina.

Ecco la lista dei comandi a disposizione:

```
\bilinear
```

per comporre carta quadrettata (millimetrata);

```
\semilogx{⟨number of decades⟩}
\semilogy{⟨number of decades⟩}
```

per comporre carta semilogaritmica, con l'asse logaritmico rispettivamente lungo le ascisse (`\semilogx`) o le ordinate (`\semilogy`) del foglio. L'argomento obbligatorio rappresenta il numero di decadi lungo l'asse logaritmico;

```
\loglog[⟨0/1⟩]{⟨x-axis decades⟩}{⟨y-axis decades⟩}
```

per comporre carta log-log. I due argomenti obbligatori rappresentano, rispettivamente, il numero di decadi lungo l'asse delle ascisse (x) e delle ordinate (y). Dato che l'area dentro la quale viene disegnata la griglia è, di default, rettangolare, la dimensione di una decade sarà, in generale, diversa lungo i due assi. Impostando il parametro opzionale a 1, le dimensioni delle decadi saranno forzate allo stesso valore lungo i due assi. L'effetto di questo parametro si può capire chiaramente confrontando la figura 4 con la figura 5, entrambe con due decadi lungo le ascisse e tre lungo le ordinate, ma con il parametro opzionale impostato a 1 nel secondo grafico;

```
\polar
```

per comporre carta polare con l'asse radiale lineare;

```
\logpolar{⟨number of decades⟩}
```

per comporre carta polare con asse radiale logaritmico. L'argomento obbligatorio rappresenta il numero di decadi lungo l'asse logaritmico. Il numero massimo di decadi è 2;

```
\smith
```

per comporre la carta di Smith. In particolare una carta di Smith per l'impedenza.

Prima di spiegare come personalizzare l'aspetto di queste carte da grafico, mostriamo un semplice codice di esempio in cui abbiamo riportato tutti i comandi visti finora.

```
\documentclass[a4paper]{graphpaper}
\begin{document}
\bilinear
\semilogx{3}
\semilogy{3}
\loglog{2}{3}
\loglog[1]{2}{3}
\polar
\logpolar{2}
\smith
\end{document}
```

Il risultato consiste in un documento di otto pagine, con una carta da grafico per ogni pagina. Nelle figure 1–8 abbiamo riportato il risultato, opportunamente riscalato, dove ogni figura corrisponde a una pagina del documento. Il rettangolo nero intorno alla figura rappresenta la dimensione del foglio A4.

2.2 Personalizzazioni

Graphpaper mette a disposizione dell'utente diversi comandi per personalizzare lo stile delle carte da grafico. Questi comandi vanno inseriti nel corpo del documento e hanno effetto su tutte le carte da grafico successive. Fa eccezione, come vedremo, il comando `\customcode`, ma il suo comportamento può essere facilmente cambiato grazie al suo argomento opzionale.

Per quanto riguarda la carta millimetrata, semilogaritmica e log-log, si hanno a disposizione questi comandi specifici per personalizzarle:

```
\setxside{⟨length⟩}
\setyside{⟨length⟩}
\setminimumdistance{⟨length⟩}
```

`\setxside` and `\setyside` permettono di specificare la dimensione del rettangolo della griglia, rispettivamente lungo l'asse delle ascisse e delle ordinate. Di default questi valori sono pari a (260 mm, 180 mm) per un foglio A4, (250 mm, 190 mm) per il formato letter, e (380 mm, 280 mm) per un foglio A3.

`\setminimumdistance` specifica, per le carte lineari e logaritmiche, la minima distanza ammessa tra due linee delle sottodivisioni. Il suo valore di default è 1 mm. Questa impostazione può anche essere usata, per esempio, per far sì che la carta bilineare abbia solo le sottodivisioni da 5 mm ma non quelle da 1 mm (cioè per ottenere una carta quadrettata da 5 mm). Per fare questo è sufficiente usare un valore superiore a 1 mm. Se si usa un valore superiore a 5 mm, si otterrà una carta bilineare con le sole linee principali, cioè una ogni cm (una carta quadrettata da 1 cm).

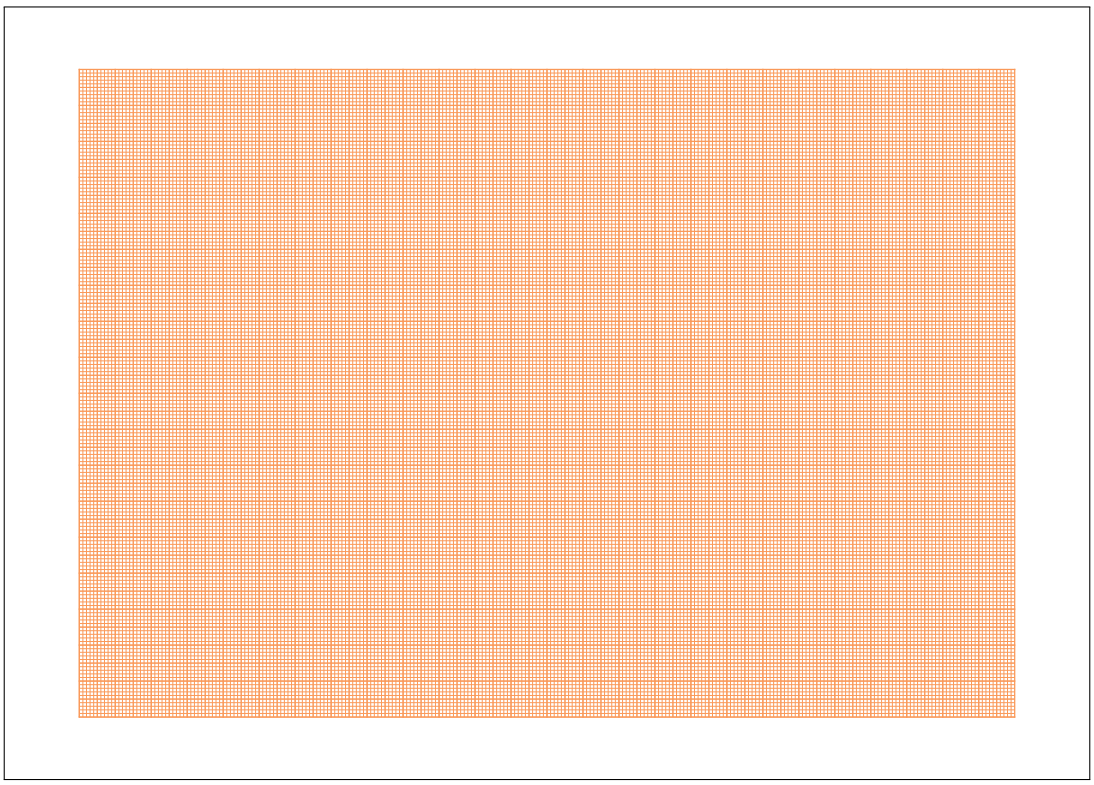


FIGURA 1: Esempio di carta da grafico realizzabile con graphpaper. Carta (bilineare) millimetrata.

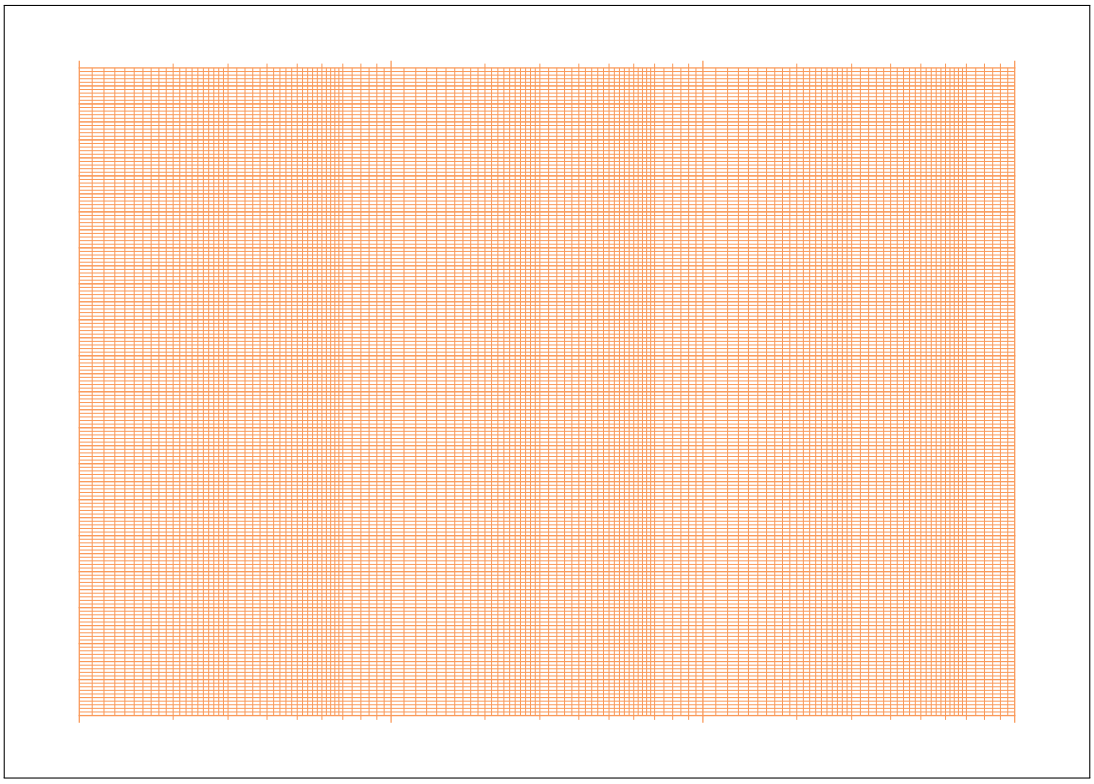


FIGURA 2: Esempio di carta da grafico realizzabile con graphpaper. Carta semilogaritmica, lineare sulle ordinate, logaritmica 3 decadi lungo le ascisse.

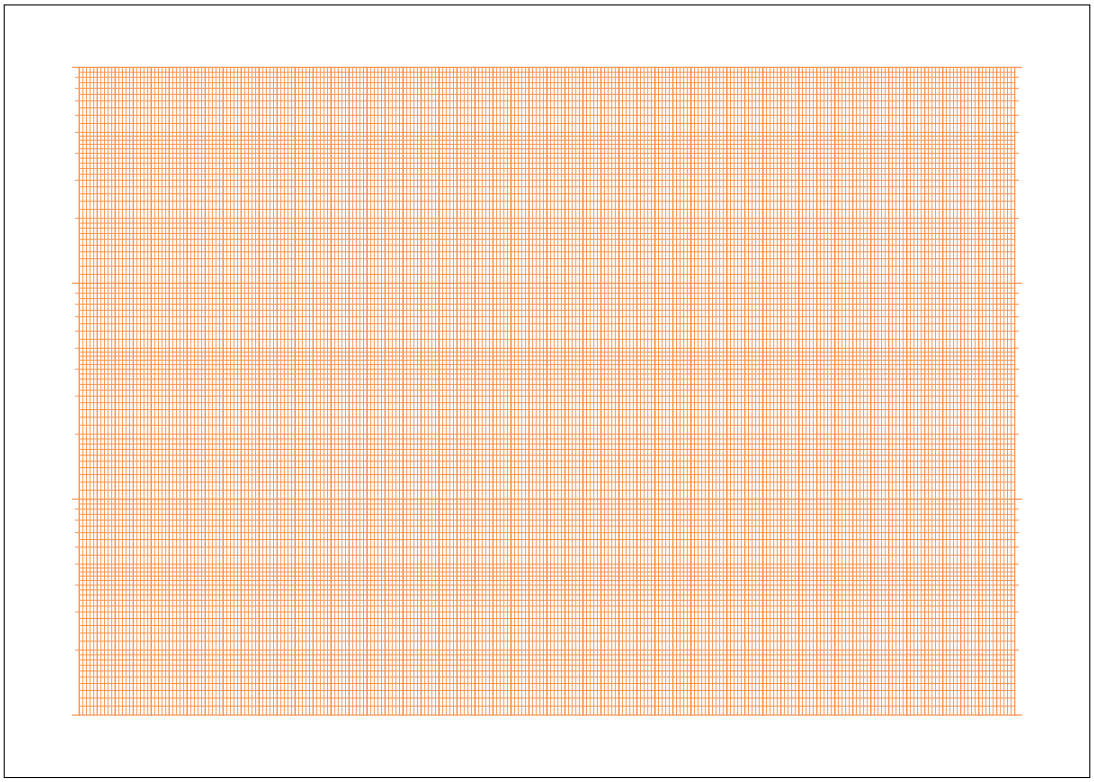


FIGURA 3: Esempio di carta da grafico realizzabile con graphpaper. Carta semilogaritmica, lineare sulle ascisse, logaritmica 3 decadi lungo le ordinate.

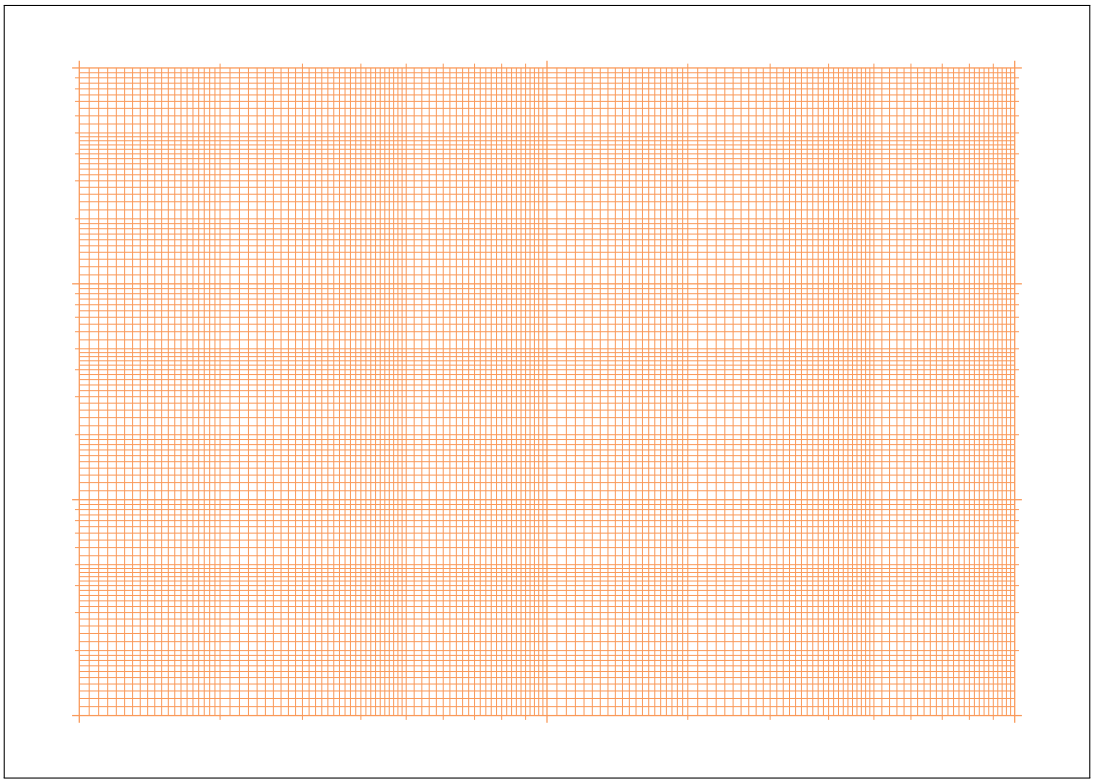


FIGURA 4: Esempio di carta da grafico realizzabile con graphpaper. Carta doppio logaritmica, 2 decadi lungo le ascisse, 3 decadi lungo le ordinate.

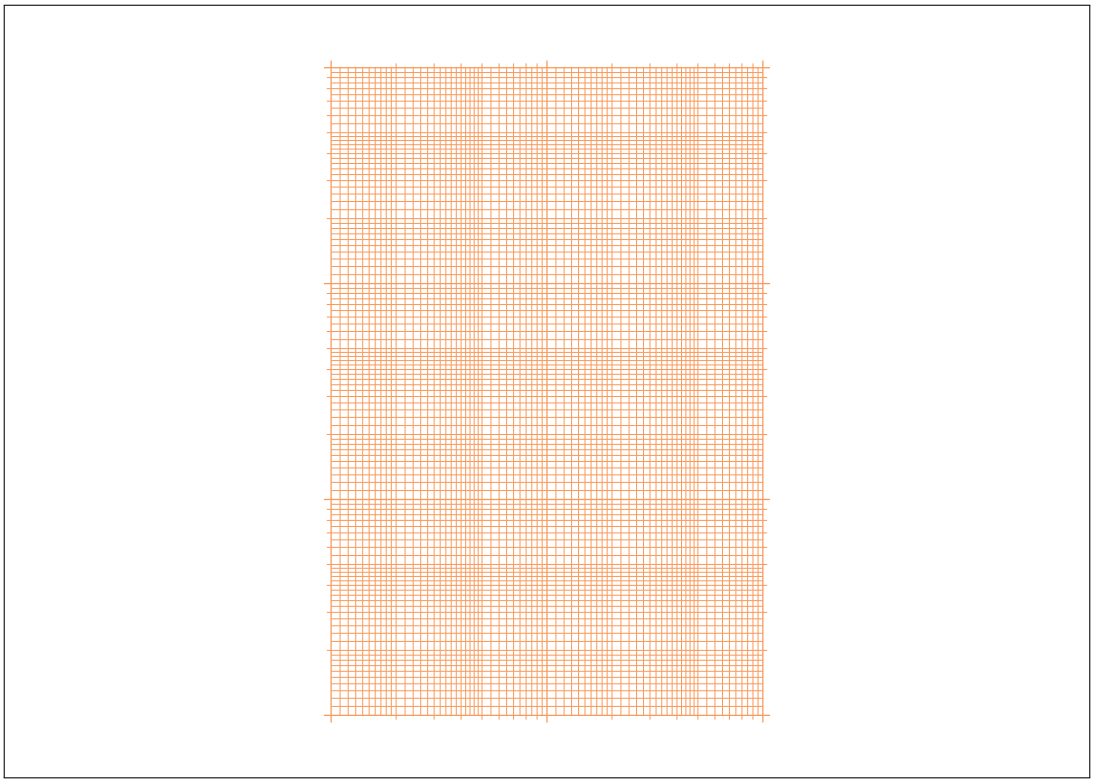


FIGURA 5: Esempio di carta da grafico realizzabile con graphpaper. Carta doppio logaritmica, 2 decadi lungo le ascisse, 3 decadi lungo le ordinate, con grandezza delle decadi uguale per i due assi.

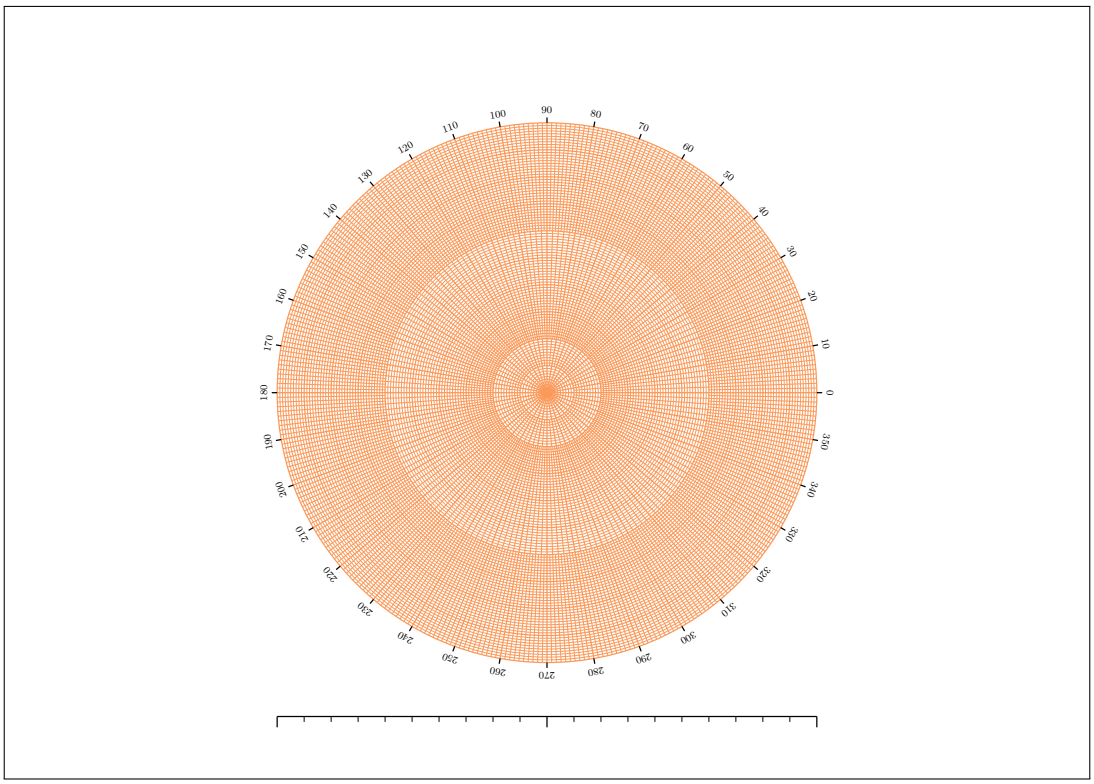


FIGURA 6: Esempio di carta da grafico realizzabile con graphpaper. Carta polare.

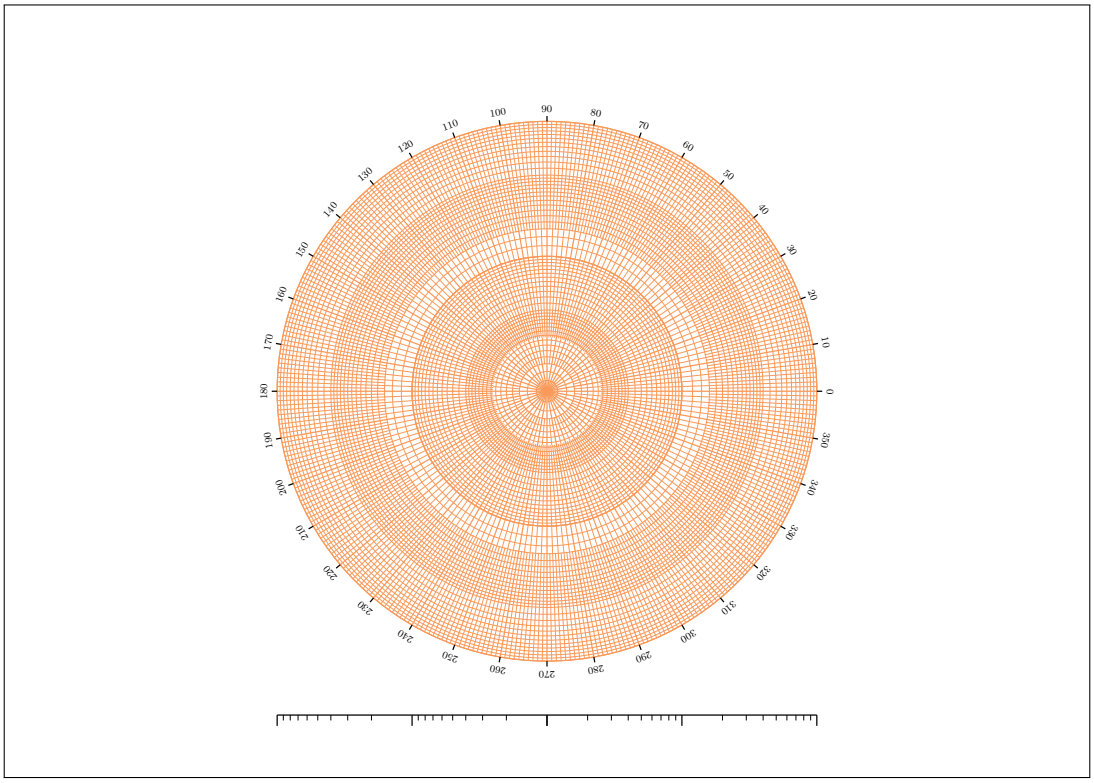


FIGURA 7: Esempio di carta da grafico realizzabile con graphpaper. Carta polare logaritmica lunga la direzione radiale (2 decadi).

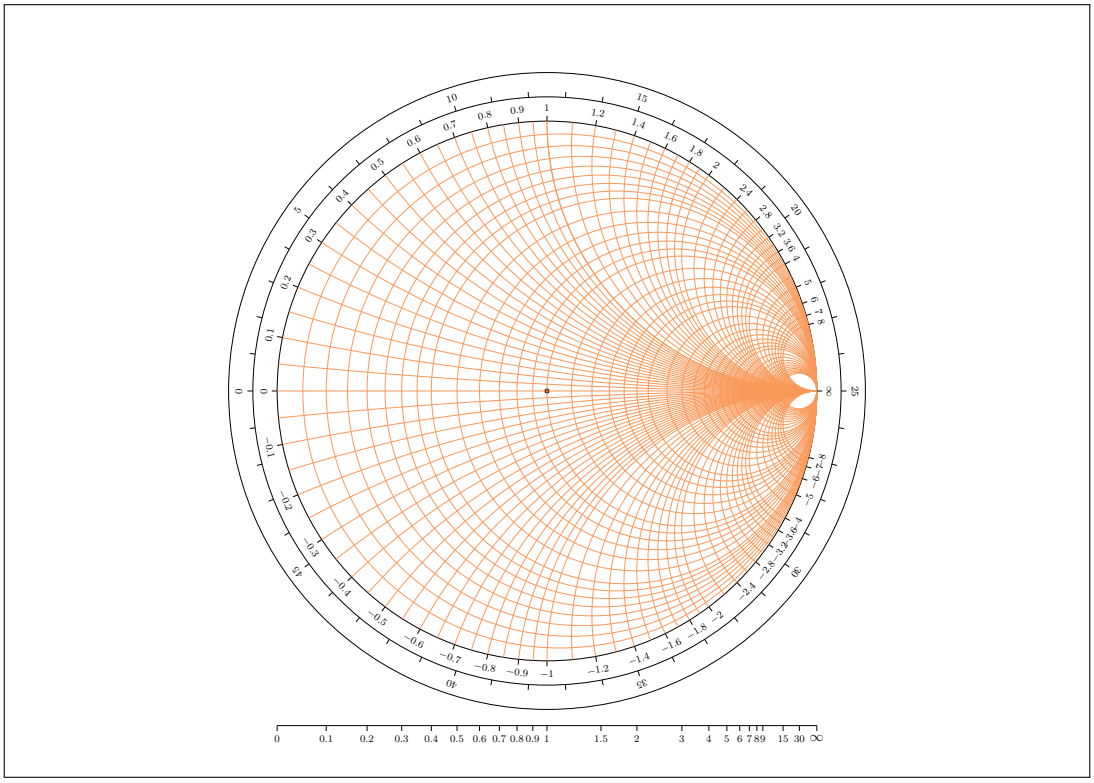


FIGURA 8: Esempio di carta da grafico realizzabile con graphpaper. Carta di Smith.

I seguenti comandi di personalizzazione sono validi invece per tutte le carte:

```
\setgridcolor{⟨color⟩}
\setmajorlinethickness{⟨length⟩}
\setmediumlinethickness{⟨length⟩}
\setminorlinethickness{⟨length⟩}
\customcode[⟨0/1⟩]{⟨picture env. commands⟩}
```

`\setgridcolor` imposta il colore delle linee. L'argomento deve essere un colore definito o predefinito dal pacchetto color o xcolor. Il colore di default è il Pantone 2011 U corrispondente all'RGB (250,153,89), il tipico colore rosso-arancio spesso usato per le carte da grafico e che si può vedere nelle figure 1–8.

`\setmajorlinethickness` permette di impostare lo spessore delle linee principali, `\setmediumlinethickness` quello delle linee medie e `\setminorlinethickness` quello delle linee sottili. I valori di default sono 1 pt, 0.6 pt, 0.25 pt, rispettivamente.

Infine, `\customcode` permette di disegnare, scrivere, inserire un logo, e così via, sopra la carta da grafico. Accetta tutti i comandi dell'ambiente picture. Il suo contenuto viene automaticamente eseguito nell'ambiente picture usato per disegnare la carta da grafico. Pertanto, per poterlo usare correttamente, è necessario conoscere dove si trova l'origine del sistema di coordinate e la sua unità base per ogni carta: nel caso delle carte lineari e logaritmiche l'origine si trova nell'angolo in basso a sinistra della griglia e la `\unitlength` è 1 mm; per le carte polari e quella di Smith l'origine si trova al centro del cerchio e la `\unitlength` è impostata a 1/140 dell'altezza del foglio di carta in uso in quel momento. L'argomento opzionale, se impostato a 0, fa in modo che la serie di comandi dati nell'argomento obbligatorio di `\customcode` non venga cancellata dopo ogni sua esecuzione.

Vediamo un esempio d'uso. Il codice che segue disegna due carte con un grafico ciascuna; una è millimetrata e l'altra semilogaritmica.

```
\documentclass[a4paper]{graphpaper}
\begin{document}
\setxside{10cm}
\setyside{10cm}
\setminimumdistance{5mm}
\setgridcolor{blue}
\setmajorlinethickness{1mm}
\setmediumlinethickness{0.6mm}
\setminorlinethickness{0.2mm}
\customcode{%
  \put(130,110){%
    \includegraphics[width=3cm]{logo.pdf}}
  \color{red}
  \segment(0,10)(60,80)}
\bilinear
\setgridcolor{red}
% Definizione logaritmo in base 10.
% 50 corrisponde alla lunghezza del lato
% logaritmico (100 mm)
% diviso 2 (il numero di decadi)
\newcommand{\Log}[1]{%
  \fpeval{ln(#1)/ln(10)*50}}
\customcode{%
  \put(130,110){%
    \includegraphics[width=3cm]{logo.pdf}}
  \color{black}
  \segment(\Log{1},10)(\Log{60},80)}
\semilogx{2}
\end{document}
```

All'inizio viene definita la grandezza della carta (10 cm × 10 cm). E questa impostazione si applica automaticamente a entrambe le carte. Viene poi definita la minima distanza delle sottodivisioni. Nel caso della carta millimetrata questo equivale a rendere la carta millimetrata una carta quadrettata con quadratini di lato 5 mm. Viene poi impostato il colore e la larghezza delle linee. Il comando `\customcode` viene usato in entrambi i casi per

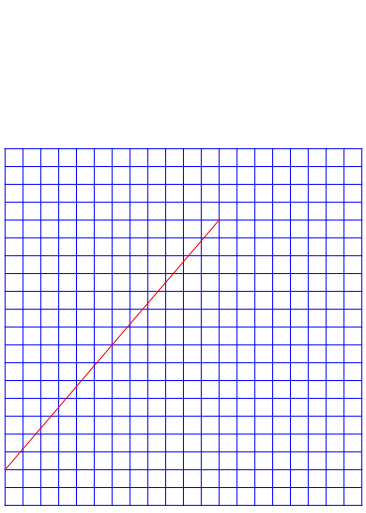


FIGURA 9: Prima pagina dell'esempio sulla personalizzazione riportato nel testo

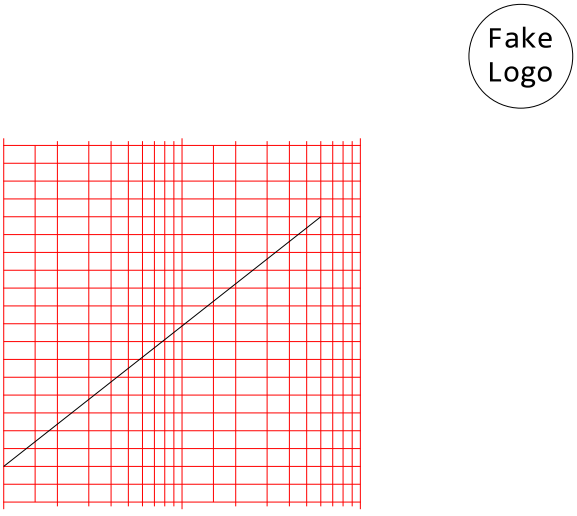


FIGURA 10: Seconda pagina dell'esempio sulla personalizzazione riportato nel testo

inserire un logo e disegnare un segmento. Il risultato è mostrato nelle figure 9 e 10, dove ogni figura corrisponde a una pagina, opportunamente riscalata e ritagliata, del file PDF risultate dopo la compilazione.

3 Conclusioni e prospettive

In questo articolo è stata presentata graphpaper, una classe LaTeX per la creazione e personalizzazione di molti tipi di carta per grafici (carta millimetrata, semilogaritmica, log-log, polare, polare logaritmica, di Smith).

Tra le future migliorie che stiamo valutando di introdurre in una prossima versione figurano: un comando per disegnare le linee e i *tick* delle carte lineari a intervalli desiderati (quindi non solo millimetrate); un comando per scrivere le etichette dei valori corrispondenti ai vari *tick* sia per la carta lineare che per quelle logaritmiche; un comando per passare da un certo valore di input al valore della coordinata necessaria all'ambiente picture per poter riportare correttamente in scala logaritmica quel valore (come nell'esempio); un comando analogo al precedente per le carte polari logaritmiche; dare a tutti i comandi di personalizzazione la possibilità di avere effetto solo fino alla prima esecuzione di un comando per la composizione di una carta.

Ci auguriamo che questo lavoro non sia solo una buona classe LaTeX per preparare carte per grafici, ma che stimoli la riflessione sull'uso di questi strumenti nell'insegnamento.

A File per stampa offset

Usando una versione preliminare di questa classe FB è riuscito a risolvere il problema che aveva posto sul forum del GUIT descritto nell'introduzione. Ha preparato tre distinti pdf: uno con carta millimetrata e due con carte semilogaritmiche, una con 2 decadi e l'altra con 3 decadi.

Per tenere bassi i costi di stampa ha deciso di stampare 5000 copie per la millimetrata e 3000 copie ciascuna per le semilogaritmiche. Dopo diversi preventivi, il costo minore è stato offerto da una tipografia con stampa offset monocolore. Questo tipo di tecnica di stampa è quella che offre anche la migliore accuratezza e precisione nello spessore e nella posizione dei tracciati. Il costo totale è stato di 0,0304 € per foglio. I fogli sono stati anche rilegati gratuitamente a caldo in blocchetti di 50 fogli ciascuno, con un cartoncino sul fondo, e facilmente staccabili. Il risultato è riportato nella figura 11.

I tre file pdf, così come ottenuti dalla compilazione con pdflatex, non erano però adatti per la stampa offset. Infatti il pdf doveva essere monocromatico (nero su bianco) e tutti i caratteri dovevano essere convertiti in tracciati. Per la monocromaticità non c'erano problemi, infatti è stato sufficiente impostare il colore delle griglie con il comando

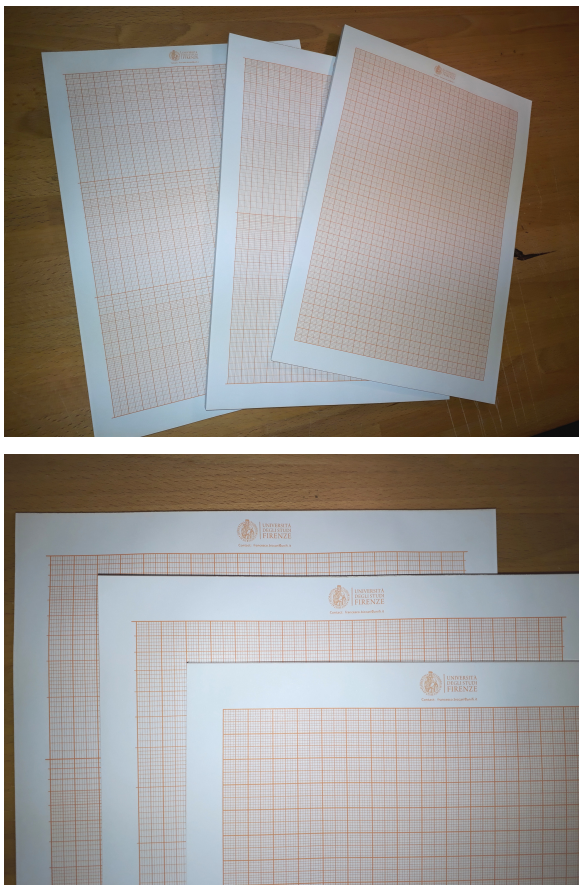


FIGURA 11: Fotografie di tre blocchetti di fogli di carta per grafici ottenuti grazie a graphpaper. La carta ha una grammatura 80 g/m^2. Ogni blocchetto contiene 50 fogli rilegati a caldo con un cartoncino sul fondo. Il colore scelto per la stampa è stato il Pantone 2011 U.

apposito di graphpaper e inoltre, fortunatamente, il logo dell'Università degli Studi di Firenze era già monocromatico. Erano invece presenti dei font, perché sotto il logo dell'Università era stato aggiunto un indirizzo email. Per convertire i font in tracciati è stato usato il seguente comando Ghostscript (sotto Windows 10, Ghostscript 9.22):

```
gswin64c.exe -q -dNOPAUSE -dBATCH -dSAFER ^
-sDEVICE=pdfwrite ^
-dCompatibilityLevel=1.5 ^
-sOutputFile="output.pdf" ^
-dNoOutputFonts ^
-dPDFSETTINGS="/prepress" ^
input.pdf
```

L'opzione che permette di convertire i font in tracciati è `-dNoOutputFonts` (vedi la documentazione Ghostscript, capitolo Vector Devices). L'opzione `-dPDFSETTINGS="/prepress"` invece lascia sostanzialmente invariata la qualità del pdf di output rispetto a quello di input (si veda la documentazione Ghostscript, capitolo Vector Devices).

Ghostscript è disponibile anche in ambiente Linux e Mac e il comando riportato sopra può essere facilmente adattato per questi sistemi (le uniche

differenze consistono nel nome dell'eseguibile e nel carattere di a capo che può essere omesso dato che qui è stato usato solo per poter scrivere il codice su più righe).

Riferimenti bibliografici

Beccari, C. e F. Biccari (2020). «The Graph-paper package». https://www.ctan.org/pkg/graphpaper. Controllato il 25 ottobre 2020.

Biccari, F. «Carta millimetrata lineare, semilogaritmica e log-log». Forum del GUIT. https://www.guitex.org/home/it/forum/5-tex-e-latex/116974-carta-millimetrata-lineare-semilogaritmica-e-log-log. 10 agosto 2019.

Feuersänger, Christian (2020). «The PGFPlots package». https://www.ctan.org/pkg/pgfplots. §5.11 della documentazione. Controllato il 26 ottobre 2020.

Wikipedia (a). «Carta di smith». https://it.wikipedia.org/wiki/Carta_di_Smith. Controllato il 25 ottobre 2020.

— (b). «Graph paper». https://en.wikipedia.org/wiki/Graph_paper. Controllato il 25 ottobre 2020.

▷ Claudio Beccari
Politecnico di Torino
claudio dot beccari at polito dot it

▷ Francesco Biccari
Università degli Studi di Firenze
francesco dot biccari at unifi dot it

TEX in church: more adventures

Paulo Roberto Massa Cereda

Abstract

This article is a continuation of the author's typographical adventures when producing material for masses and church-related activities.

Sommario

Questo articolo è il seguito delle avventure tipografiche che l'autore ha sperimentato nel produrre materiale per la messa e altre attività ecclesiastiche.

1 Introduction

Using TEX and friends beyond the academic scope is not too unusual as it sounds, specially in tasks in which typography plays an important role. However, the literature is a bit sparse when covering such scenarios and looking for potential opportunities.

In a recent TUGboat article (Cereda, 2020), I described how I use TEX for producing material for masses and church-related activities. In that particular document, I covered two topics, songsheets and song booklets, describing how TEX and friends could provide an efficient and fluent work flow, as well as producing beautiful materials. The reception was surprisingly positive. Now, this article for the Italian community can be read as a natural extension of my previous contribution, covering other areas of the church. The original advice stands: there is yet plenty of room to cover!

2 Books

I tend to not dissociate design and content when reading a book, as they offer an unique experience to the reader. If the book is of a spiritual nature (e.g, the life of a saint[1]), things get more intricate: as in sacred music, the text is food for the mind and the visual aspects are food for the heart.

Recently, good book projects can be found from local Catholic publishers. Not just the paper quality and cover artwork are well chosen, the entirety of the typographical elements is finely crafted. This offers a very positive and pleasant experience, as well as an authoritative impact: mind, heart and soul are invited to harmony.

In 2015, I had the opportunity to typeset a small songbook for our parish. That was my first contact with an actual print shop, with a circulation of 200 copies. I had to design everything from ground up, from cover to internal elements. Of course, in hindsight, I would change a couple of (if not several) things if I had the chance; the use of TEX, however, helped me understand the importance of good typography. Figure 1 illustrates two pages of the book (songs and prayer).

This particular project was challenging: I had to fit 400 songs inside the book without going over 100 pages total. Also, the text had to be readable by people in a comprehensive age range, from children to elders. I had to work with at least 6 different layouts until I found a reasonable compromise solution. I used memoir as class, followed by some important packages such as multicol and tcolorbox, as a means to produce the final result.

One interesting bit worth mentioning is the inclusion of `\textls`, used for letterspacing shorter pieces of text, alongside with `\scshape` to typeset the words of the Consecration in the Eucharistic Prayers, as a means to provide emphasis:

```
\textls{\scshape Hoc est enim
                 Corpus meum.}
```

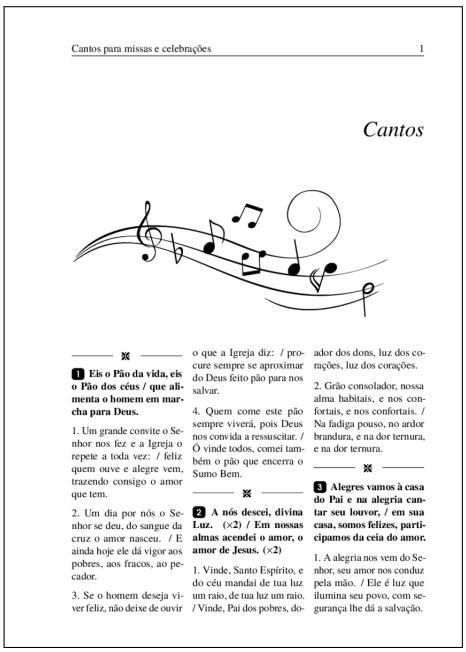
In 2018, I decided to typeset my own songbook. This time, I had only a few constraints and was able to include 900 songs in 160 pages total. I also had more space available for content, so I decided to keep the existing layout with minor tweaks and a brand new font. Figure 2 illustrates two pages of the book (songs).

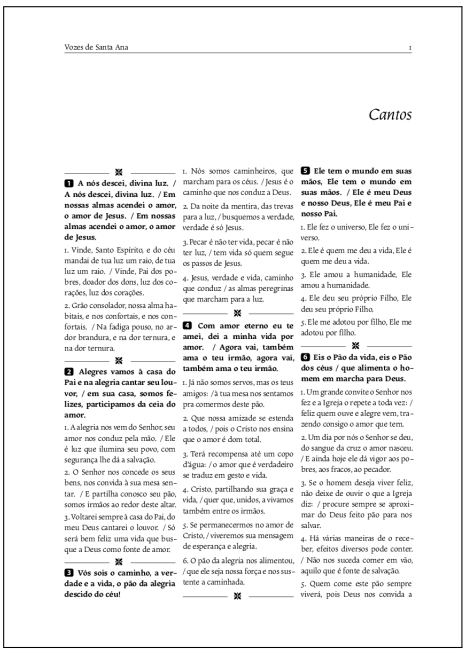
As the previous project, I used memoir as class, followed by some important packages such as multicol, tcolorbox and fontspec. This was also my first project using XeLaTeX as engine.

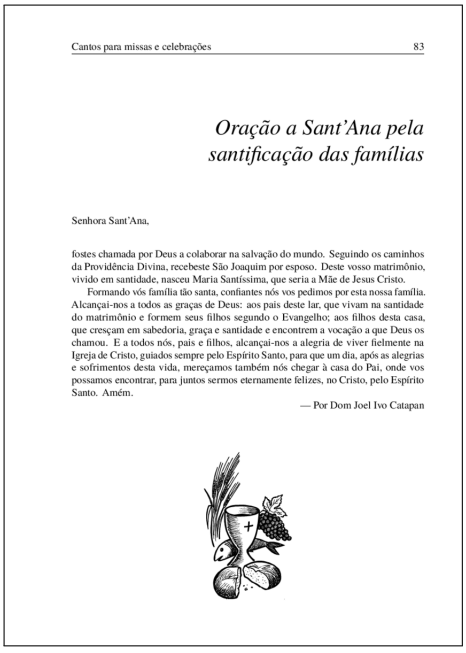
I also used imakeidx for song indexing. However, I had to patch it, as songs had to be referenced by their numbers and not by the pages in which they were located in the document. Thanks to etoolbox, the patch was straightforward:

```
\makeatletter
\patchcmd{\@wrindex}{\thepage}
{\thesongscounter}{}{}%
\AtBeginDocument{%
  \patchcmd{\imki@putindex}
    {\immediate}
    {\ifimki@disableautomatic
    \else\immediate}
    {}{}%
  \patchcmd{\imki@putindex}
    {\endcsname}
    {\endcsname\fi}
```

1. I am currently reading a book about the life of Saint Alphonsus Maria de Liguori, which is fascinating.







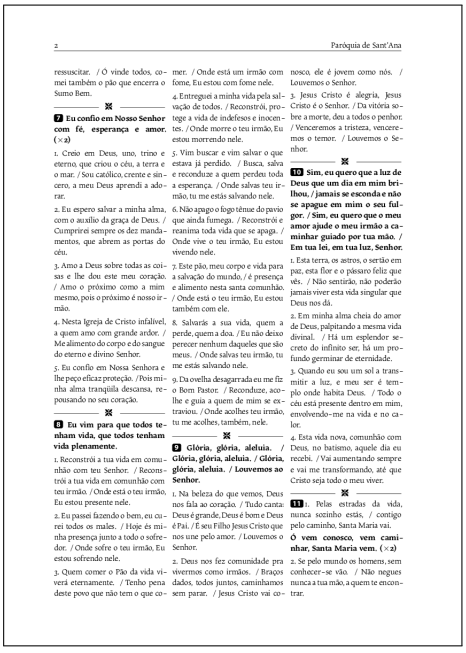


FIGURE 1: My first songbook typesetting, in 2015.

FIGURE 2: My second songbook typesetting, in 2018.

```
    {}{}%
}
\makeatother
```

In 2019, I collected songsheets from the Taizé community, an ecumenical Christian monastic fraternity in Taizé, France, and compiled them into a personal songbook as well. Figure 3 illustrates two pages of this book (cover and index).

Again, I decided to use memoir as class and packages such as tcolorbox, fontspec and imakeidx for graphical elements, font handling and index, respectively. As reference, I used the following custom style for the index:

```
headings_flag 1
heading_prefix "\\letra{"
heading_suffix "}\\nopagebreak\n"
item_0 "\n\\item "
delim_0 "\\nobreak\\dotfill "
```

In this particular style, `\letra` is a wrapper around a custom tcolorbox environment which helped me achieve the effect of boxed letters as section headings in the printed index.

Songsheets were included as images, so I had an external script to calculate the box widths and set up the dimensions in my document accordingly. I also used XeLaTeX as engine. The result was quite interesting, as I managed to automate almost every step of song inclusion.


Paróquia de Sant'Ana
Diocese de Limeira

Cânticos de Taizé
Coletânea de cânticos meditativos

Paulo Roberto Massa Cereda

Analândia
2019

Cânticos de Taizé iii

Índice alfabético

A
Aber du weisst den Weg für mich 15
Ad te Jesu Christe 12
Adoramus te Christe 3
Adoramus te O Christe 13
Alleluia 9
Alleluia (Kiev) 14
Alleluia 10 17
Alleluia 11 11
Alleluia 16 18
Alleluia 17 6
Alleluia 18 5
Alleluia 20 4
Alleluia 21 2
Alleluia 22 7
Alleluia 4 1
Alleluia 7 19
Alleluia 8 10
Atme in uns 8

B
Beati voi poveri 26
Behüte mich Gott 20
Bendigo al Señor 23
Benedictus (canon) 24
Benedictus Dominus Deus 21
Bleib mit deiner Gnade 27
Bleibet hier 22
Bless the Lord 29
Bogoroditse Dievo i 32
Bogoroditse Dievo 2 25
Bonum est confidere 30
Bénissez le Seigneur 31
Bóg jest miłością 28

C
Cantarei ao Senhor 37
Cantate canticum novum 39
Christe Salvator 38
Christe, lux mundi 35
Christus resurrexit 40
Confitemini Domino 36
Crucem tuam 33
C'est toi ma lampe 34

D
Da pacem cordium 41
Da pacem in diebus 42
Dans nos obscurités 44
De noche 43
Dieu ne peut que donner 46
Dominus spiritus est 48
Dona la pace Signore 47
Dona nobis pacem 45

E
Eat this bread 51
El alma que anda en amor 52
El Senyor 54
Esprit consolateur 50
Exaudi orationem meam 49
Exomologisthe to Kyrio 53

F
Fiez-vous en Lui 56
Frieden, Frieden 57

G
Gib mich ganz zu eigen dir 58
Gloria et in terra pax 60
Gloria, gloria 61
Grande est ta bonté 59

H
Heureux qui s'abandonne à toi 62

FIGURE 3: My third songbook typesetting, in 2019.

Paper size

Some decisions were result of years of experimentation, as well as trial and error sessions. The first choice to make was about the paper size. A4 was pretty much the norm around here, but I was concerned about how uncomfortable and clumsy it was for people to hold an A4 paper sheet during masses and other church-related activities. I was also suspicious that the larger the paper, the quicker it would be damaged and had to be replaced.

At a certain moment in time, I decided to try A5 for a change. People were quite reluctant on holding a paper different than the traditional A4 size. It took a couple of weeks. Soon, I saw the potential on using A5, not to mention the possibility of saving sheets, since I could print two A5 pages (or four, considering the two sides, as seen in Figure 4) in just one A4 page. In a humble parish like ours, this would be a most welcoming benefit.


Quase Paróquia
Santo Mario e Santo Paulo
Assembleia Paroquial 2018
2º Domingo do Advento
9 de Dezembro de 2018
"Prepara o caminho do Senhor, endireita suas veredas" – Lucas 3:4
+ Ação de graças
+ Final
+ Batizo aspersão
+ Entrada
+ Ato penitencial
+ Aclamação
+ Apresentação das ofertas
+ Salmo responsorial
+ Santo
+ Doxologia
+ Cordeiro de Deus
+ Comunhão

FIGURE 4: Booklet sample from 2018.

So I found a paper size that would be ideal for our parish. It did not take long for me to start experimenting booklet formats as well. It was a success. Thanks to the pdfpages package, which provides support for booklets, I was able to organise A5 pages together such that I could just print and fold A4 sheets without the need of cutting:

```
\includepdf[pages={4,1,2,3},
                    nup=2x1]{booklet}
```

I even acquired a special stapler to help me staple in the middle of an A4 page when we need to produce an A5 booklet with more than four pages, as seen in Figure 5.

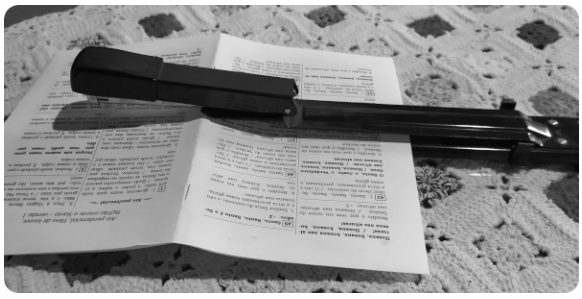


FIGURE 5: My special stapler.

From now on, A5 is our paper size of choice for books and booklets, as it seems the most reasonable

balance between cost and benefit for our parish. I have plans for A6 booklets as well (specially devotional material), but that might take a while.

Fonts

Fonts do play an important role in producing church-related material. In a way, the type choice might even convey an impression of credibility and assurance (or lack thereof, in the case of unfortunate decisions[2]).

For some reason[3], certain popular publishes are increasingly adopting the use of sans serif fonts. While these documents do not look that bad at all, I still favour the use of serif fonts as the norm for printed material. Occasionally, I might rely on a sans serif font for a section title, but even so the font has to be carefully chosen.

How do I choose a font? The answer is not too simple. For starters, I need to know the target public, so I can direct my efforts towards enhancing their reading experience. For instance, when preparing an order of mass, I have to assess the peculiarities of this scenario:

1. There are people in a comprehensive age range, from children to elders, so I need to ensure the text is readable. I tend to opt for rounder and wider fonts, so Bookman-based typefaces are an excellent choice. It is also possible to experiment with microtype as a means to tweak character protrusion and font expansion.

2. The church building does not provide too much natural light during masses, so the font size has to be sufficient to be spotted in a low light environment. In this regard, due to its stylistic nature, Bookman seems to be an interesting choice again, as it is readable even at smaller sizes.

3. Considering the use of an A5 paper size and the area available for the actual content, how should I deal with text alignment and hyphenation rules? This is way trickier than it sounds: there has to be a balance between good typography and readability.

There is a lot yet to consider, including the use of bold, italics and other styles throughout the text. Are they really necessary? The human eye is very receptive to differences in a text. However, a text should not exaggerate on such emphasis, as they could (unintentionally) distract the reader from an actual text comprehension and the message would be lost. As my mother usually says, a text should not contain too much information coming from styles or shapes. Less is more.

When preparing material for children, fonts with irregular shapes like Comic Sans are actually welcome. I am looking forward to an opportunity to use an interesting font named Open Dyslexic, created to increase readability for readers with dyslexia. According to the author, letters have heavy weighted bottoms to indicate direction, and the unique shapes of each letter can help prevent confusion through flipping and swapping.

There is another interesting font project named Luciole, which might explain the increasing use of sans serif by certain publishers. The typeface is developed explicitly for visually impaired people. According to the development team, the font adheres to several design criteria to provide the best possible reading experience for the visually impaired. It is indeed worth giving a try, specially in a context in which people could certainly benefit from such font features.

If you are curious about the fonts I typically use in a project, here is a non-exhaustive list: Bookman, Karol, Calendas Plus, Arno Pro, Calluna, Source Serif Pro, Lucida, Elstob, EB Garamond, Alegreya, Sabon Next, Walbaum and Quire Sans. This list contains proprietary and open source fonts.

Layout

Since I had some constraints when working on these projects (specially number of pages as a means to save printing), I decided to experiment some minimalist designs. I then opted for smaller titles and graphical elements unobtrusively disposed in the page. The feedback was positive. I also refrained myself from using too much colour (if any), so the majority of my material is monochromatic.

For the layout itself, I found out columns could be an interesting approach for taking advantage of the space available in the page in a clever way (specially with balanced columns). The question was, how many columns should I use? It depends, again, on several aspects.

For smaller font sizes, three columns seem to be the most sensible approach, as seen in Figures 1 and 2. For larger fonts or for certain scenarios, two columns seem to be the norm. Figure 6 illustrates the use of two columns in a song booklet for masses.

Personally, I always find useful to prepare document proofs, i.e, print some samples and ask for feedback during production. In here, my parents work as my layout reviewers. Everything is put under scrutiny: text, layout, paper size, printing quality, font size and style, and other graphical elements. The feedback, however, does not necessarily has to be of a technical nature. Even the simplest suggestion or criticism (or the lack thereof) contributes to the project typesetting. The more the document looks fluid and pleasant to the eye, the better impact on its potential readers.

2. I still find some parishes using fonts like Comic Sans for bulletins and devotionals. While I do not hold anything against this font family and its derivatives, I strongly believe they should be used with absolute great care and attention in very specific scenarios, as seen later in this article.

3. Actually, I have an hypothesis on this particular topic, which I expand a bit later on in this section.


Campanha da Fraternidade 2020

1 1. Deus de amor e de ternura, contemplamos / este mundo tão bonito que nos deste. / Desse Dom, fonte da vida, recordamos: / Cuidadores, guardiões tu nos fizeste.

Peregrinos, aprendemos nesta estrada / o que o bom samaritano ensinou: / ao passar por uma vida ameaçada, / ele a viu, compadeceu-se e cuidou.

2. Toda vida é um presente e é sagrada, / seja humana, vegetal ou animal. / É pra sempre ser cuidada e respeitada, / desde o início até seu termo natural.

3. Tua glória é o homem vivo, Deus da Vida, / ver felizes os teus filhos, tuas filhas, / é a justiça para todos, sem medida, / é formarmos, no amor, bela Família.

4. Mata a vida o vírus torpe da ganância, / da violência, da mentira e da ambição. / Mas também o preconceito, a intolerância. / O caminho é a justiça e conversão.

2 Acolhe, ó Deus, o nosso canto! / Escuta, pois, nosso clamor. / Vê quanta dor, quanto pranto / que o nosso pecado provocou. / Por teu amor, tem misericórdia! / Por teu amor, tem misericórdia!

1. Voltai, agora, para mim / com coração arrependido. / Rasgai, não as vestes, / mas o coração, diz o Senhor!

2. Se o ímpio se arrepender / dos pecados cometidos / e praticar a

FIGURE 6: Song booklet from 2020.

Columns are handled by the multicol package. I usually rely on balanced columns (which is the default behaviour when using the multicols environment), for mainly two reasons. The first is of course aesthetic, as text and other graphical elements are equally distributed in all columns. The human eye is keen on symmetry. The second reason is actually strategical, as the remaining space could be used for an additional content. The multicol package offers several features for multiple columns, including tweaks on its inner workings and column separators.

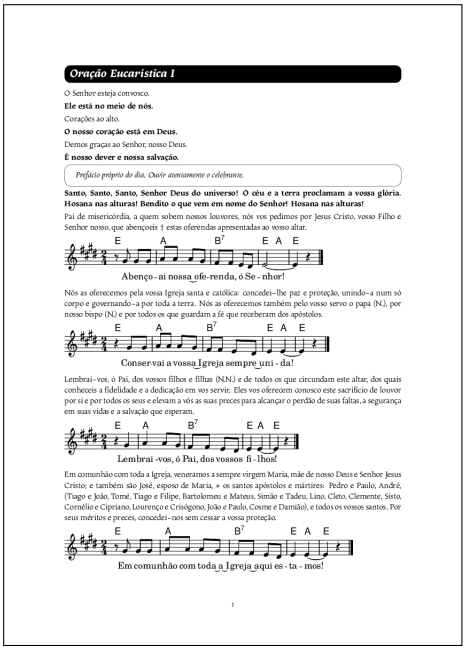
3 Assorted materials

Besides books and booklets, I use TEX and friends to produce other church-related materials. This section aims at presenting some of them, from several areas and topics of interest.

Figure 7 presents (parts of) the Eucharistic Prayer I (also known as the Roman Canon), typeset with LuaLATEX and Lilypond thanks to the lyluatex package. This package also allows the use of system fonts within Lilypond; however, we decided to keep the default sans serif font for the lyrics, defined by the engraver. It is worth mentioning, as a trivia, that the Brazilian edition of the Eucharistic Prayers has responses between sections (approved by the Vatican).

The use of the lyluatex package is quite straightforward. Keep in mind that the package requires LuaLATEX with shell escape enabled, as it invokes Lilypond under the hood. A simple macro pointing to the Lilypond file to be typeset suffices:

```
\lilypondfile{sheets/amen.ly}
```


Oração Eucarística I

O Senhor esteja convosco.
Ele está no meio de nós.
Corações ao alto.
O nosso coração está em Deus.
Demos graças ao Senhor, nosso Deus.
É nosso dever e nossa salvação.

Prefácio próprio do dia. Ouvir atentamente o celebrante.

Santo, Santo, Santo, Senhor Deus do universo! O céu e a terra proclamam a vossa glória. Hosana nas alturas! Bendito o que vem em nome do Senhor! Hosana nas alturas!

Pai de misericórdia, a quem sobem nossos louvores, nós vos pedimos por Jesus Cristo, vosso Filho e Senhor nosso, que abençoeis † estas oferendas apresentadas ao vosso altar.

E A B7 E A E
Abenço - ai nossa ofe-renda, ó Se - nhor!

Nós as oferecemos pela vossa Igreja santa e católica: concedei-lhe paz e proteção, unindo-a num só corpo e governando-a por toda a terra. Nós as oferecemos também pelo vosso servo o papa (N.), por nosso bispo (N.) e por todos os que guardam a fé que receberam dos apóstolos.

E A B7 E A E
Conser-vai a vossa Igreja sempre uni - da!

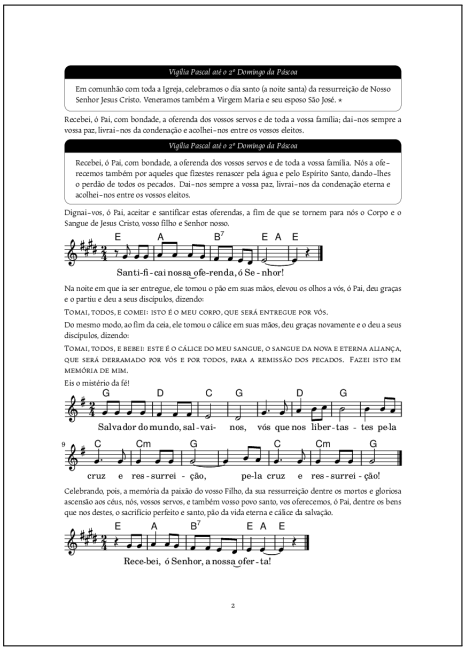
Lembrai-vos, ó Pai, dos vossos filhos e filhas (N.N.) e de todos os que circundam este altar, dos quais conheceis a fidelidade e a dedicação em vos servir. Eles vos oferecem convosco este sacrifício de louvor por si e por todos os seus e elevam a vós as suas preces para alcançar o perdão de suas faltas, a segurança em suas vidas e a salvação que esperam.

E A B7 E A E
Lembrai -vos, ó Pai, dos vossos fi - lhos!

Em comunhão com toda a Igreja, veneramos a sempre virgem Maria, mãe de nosso Deus e Senhor Jesus Cristo; e também são José, esposo de Maria, ∗ os santos apóstolos e mártires: Pedro e Paulo, André, (Tiago e João, Tomé, Tiago e Filipe, Bartolomeu e Mateus, Simão e Tadeu, Lino, Cleto, Clemente, Sisto, Cornélio e Cipriano, Lourenço e Crisógono, João e Paulo, Cosme e Damião), e todos os vossos santos. Por seus méritos e preces, concedei-nos sem cessar a vossa proteção.

E A B7 E A E
Em comunhão com toda a Igreja aqui es - ta - mos!

1


Vigília Pascal até o 2º Domingo da Páscoa

Em comunhão com toda a Igreja, celebramos o dia santo (a noite santa) da ressurreição de Nosso Senhor Jesus Cristo. Veneramos também a Virgem Maria e seu esposo São José. ∗

Recebei, ó Pai, com bondade, a oferenda dos vossos servos e de toda a vossa família; dai-nos sempre a vossa paz, livrai-nos da condenação e acolhei-nos entre os vossos eleitos.

Vigília Pascal até o 2º Domingo da Páscoa

Recebei, ó Pai, com bondade, a oferenda dos vossos servos e de toda a vossa família. Nós a oferecemos também por aqueles que fizestes renascer pela água e pelo Espírito Santo, dando-lhes o perdão de todos os pecados. Dai-nos sempre a vossa paz, livrai-nos da condenação eterna e acolhei-nos entre os vossos eleitos.

Dignai-vos, ó Pai, aceitar e santificar estas oferendas, a fim de que se tornem para nós o Corpo e o Sangue de Jesus Cristo, vosso filho e Senhor nosso.

E A B7 E A E
Santi-fi - cai nossa ofe-renda, ó Se - nhor!

Na noite em que ia ser entregue, ele tomou o pão em suas mãos, elevou os olhos a vós, ó Pai, deu graças e o partiu e deu a seus discípulos, dizendo:

TOMAI, TODOS, E COMEI: ISTO É O MEU CORPO, QUE SERÁ ENTREGUE POR VÓS.

Do mesmo modo, ao fim da ceia, ele tomou o cálice em suas mãos, deu graças novamente e o deu a seus discípulos, dizendo:

TOMAI, TODOS, E BEBEI: ESTE É O CÁLICE DO MEU SANGUE, O SANGUE DA NOVA E ETERNA ALIANÇA, QUE SERÁ DERRAMADO POR VÓS E POR TODOS, PARA A REMISSÃO DOS PECADOS. FAZEI ISTO EM MEMÓRIA DE MIM.

Eis o mistério da fé!

G D C G D G
Salva-dor do mundo, sal-vai- nos, vós que nos liber-tas - tes pe-la

9 C Cm G C Cm G
cruz e res - surrei - ção, pe-la cruz e res - surrei - ção!

Celebrando, pois, a memória da paixão do vosso Filho, da sua ressurreição dentre os mortos e gloriosa ascensão aos céus, nós, vossos servos, e também vosso povo santo, vos oferecemos, ó Pai, dentre os bens que nos destes, o sacrifício perfeito e santo, pão da vida eterna e cálice da salvação.

E A B7 E A E
Rece-bei, ó Senhor, a nossa ofer - ta!

2

FIGURE 7: Eucharistic Prayer I (incomplete).

Figure 8 presents a monthly bulletin. I write and typeset this material for the liturgy pastoral in which I work as a volunteer. This bulletin is typeset in A5 and distributed at the end of the masses. It is a very short bulletin (typically two pages), so we have to manually cut the A4 paper in half with the help of a manual paper cutting machine. We produce no more than 40 copies per edition. Occasionally, we provide copies of past editions on demand, according to certain requests.

The bulletin contains reports, news, film and book recommendations, as well as puzzles and other recreational activities. For instance, I use the amusing soup package for generating an alphabet soup, a type of puzzle consisting of a grid of letters for word search:

A clave

Boletim informativo da Pastoral do Canto Litúrgico da Paróquia de Sant'Ana

Número 9 — Salve Maria! — 04/09/2019

Procura-se um padre

— Paulo

Quando criança, nas vezes em que acompanhava meu pai ao cemitério de nossa cidade, por ocasião de algum acontecimento especial (como a missa celebrada no dia de Finados, por exemplo), lembro-me das histórias que ele me contava sobre as pessoas que ali encontravam-se sepultadas. Ainda que eu não os tivesse conhecido em vida, podia trazê-los à memória por meio dessas histórias, algumas mais tristes, outras mais felizes. Praticamente todos, sem qualquer distinção, tinham uma menção honrosa. Entretanto, havia uma exceção.

Lembro que meu pai me contava sobre um padre, de nome Matheus Pugliese, que estava sepultado no cemitério de Analândia. Mas, para minha tristeza, não havia história. Há muito havia partido para a casa paterna. Hoje, o que lhe resta é apenas seu singelo túmulo, logo na entrada do cemitério, ao lado direito. Bastante castigado pelo tempo e pelo descaso, o túmulo do então desconhecido padre Matheus bravamente resiste ao teste do tempo.

É claro que a ausência da história da vida, da obra e do legado de um de nossos padres me deixou intrigado na época. Aliás,

1

cordou a figura doce e cheia de mansidão de um pontífice forte na fé, firme nos princípios, mas sempre disponível para a acolhida e o sorriso.

"Mestre de verdade e catequista apaixonado, recordava a todos os crentes, com a fascinante simplicidade que o caracterizava, o compromisso e a alegria da evangelização, sublinhando a beleza do amor cristão, única força capaz de depor a violência e de construir uma humanidade mais fraterna", afirmou Bento XVI.

"Mestre de verdade e catequista apaixonado."

O Papa ainda considerou que, no amor e na confiança na Virgem, se encontra o segredo de uma serenidade cotidiana e de um compromisso concreto a favor da paz no mundo, que caracterizou João Paulo I.

* * *

O filme *João Paulo I: O sorriso de Deus* está disponível gratuitamente no YouTube, com áudio e legendas em Português, através do seguinte link: https://youtu.be/dXJABrFUp7Q

S	G	C	O	N	F	I	A	N	Ç	A	O			
N	I	X	J	R	H	P	V	C	E	Q	S			
O	A	M	O	R	Q	J	I	D	I	A	O			
O	H	O	P	Q	U	R	A	D	P	Z	R			
E	H	I	Q	L	T	D	R	U	O	O	R			
S	M	L	N	Q	I	X	L	M	B	O	I			
X	H	O	A	N	H	C	P	Z	R	N	S			
O	P	U	E	M	P	L	I	U	E	L	O			
F	X	R	Z	C	Q	B	L	D	Z	R	B			
T	E	J	A	L	E	G	R	I	A	S	C			
S	J	U	R	F	F	U	F	D	T	D	Z			
B	F	I	R	M	E	Z	A	S	M	T	E			

Caça-palavras

Encontre, na sopa de letrinhas ao lado, as seguintes palavras que marcaram a vida de Dom Albino Luciani, o Papa João Paulo I:

AMOR	POBREZA
FIRMEZA	CONFIANÇA
ALEGRIA	SERENIDADE
SORRISO	SIMPLICIDADE

Expediente

A clave — Boletim informativo da Pastoral do Canto Litúrgico, Paróquia de Sant'Ana, Analândia, São Paulo, Diocese de Limeira, Região Norte. Tiragem: 50

Os ensaios para missas são realizados semanalmente, às sextas-feiras, a partir das 19:30h, na Igreja Matriz. Venha participar da nossa missão!

4

FIGURE 8: A monthly bulletin (two editions).

```
% preamble
\usepackage{soup}

[...]

% document body
\begin{Alphabetsoup}[6][6][\sffamily]
\hideinsoup{1}{1}{down}{G,U,I,T}
\hideinsoup{1}{3}{right}{I,T,A,L,I,A}
\hideinsoup{1}{6}{upright}{C,A,R,L,A}
\end{Alphabetsoup}
```

Figure 9 presents a script of prayers for the Holy Hour before the Blessed Sacrament. This project was quite interesting, as it had sections with songs

Hora Santa 2018

Pastoral do Canto Litúrgico

Introdução

Sejam bem-vindos, irmãos e irmãs, a este momento de adoração. Como filhos amados de Deus, nos reunimos para adorar o Cristo Eucarístico, que fortalece a nossa fé, reanima a nossa esperança e nos faz crescer na misericórdia e no amor. Iniciemos:

Em nome do Pai, em nome do Filho, em nome do Espírito Santo, estamos aqui! (×2)

Acolhamos Jesus Eucarístico, cantando cheios de fé:

1. Bendito, louvado seja (×2) / o Santíssimo Sacramento. (×2)
2. Os anjos, todos os anjos (×2) / louvem a Deus para sempre, amém. (×2)
3. Os santos, todos os santos (×2) / louvem a Deus para sempre, amém. (×2)
4. Os pecados, todos os pecados (×2) / louvem a Deus para sempre, amém. (×2)

Graças e louvores sejam dados a todo momento.

Ao Santíssimo e Diviníssimo Sacramento.

Graças e louvores sejam dados a todo momento.

Ao Santíssimo e Diviníssimo Sacramento.

Graças e louvores sejam dados a todo momento.

Ao Santíssimo e Diviníssimo Sacramento.

Glória ao Pai, ao Filho e ao Espírito Santo.

Como era no princípio, agora e sempre. Amém!

Jesus, manso e humilde de coração.

Fazei o nosso coração semelhante ao vosso!

Oração

Ó dulcíssimo Jesus, que, escondido na Eucaristia, ouve piedosamente nossas súplicas humildes, para apresentá-las no trono do Altíssimo, aceite bondosamente agora os desejos ardentes de nossos corações. Iluminai a nossa mente, sustentai a nossa vontade, revigorai a nossa constância. E acendei em nossos corações o fogo de um entusiasmo santo, para que,superando nossa pequenez e vencendo todas as dificuldades, saibamos render-te uma adoração menos indigna de sua grandeza e majestade, e mais adequada aos nossos anseios e nossos santos desejos.

Amém.

Na presença de Cristo

Com os olhos fixos no Santíssimo Sacramento vamos colocar nossas vidas e nossas famílias aos pés de Cristo, Nosso Senhor e Salvador e clamar pela sua misericórdia. A misericórdia de Deus é como uma porta sempre aberta que nos convida a experimentar o seu amor que consola, perdoa e dá esperança. Diante de vós, entregamos nossa vida.

Jesus Ressuscitado, presente na Eucaristia, enche-nos de tua alegria e graça da ressurreição.

A Ressurreição de Jesus é o grande gesto de amor e misericórdia do Pai. Todas as vezes que participamos da Eucaristia fazemos memória, reconhecendo seu amor por todos nós.

Jesus Ressuscitado, presente na Eucaristia, enche-nos de tua alegria e graça da ressurreição.

Jesus Cristo é o rosto da misericórdia, torna-a visível, através dos seus relacionamentos, dos seus gestos e sinais, sobretudo com os pobres e necessitados.

Jesus Ressuscitado, presente na Eucaristia, enche-nos de tua alegria e graça da ressurreição.

Jesus sentiu compaixão da multidão, que era como ovelha sem pastor, curou os doentes e saciou o povo faminto. Enfim, tudo Nele é misericórdia.

Jesus Ressuscitado, presente na Eucaristia, enche-nos de tua alegria e graça da ressurreição.

Ter misericórdia é possuir o mesmo sentimento de Jesus, fazendo o bem especialmente aos mais pobres, abandonados e desesperançados. Para isso, é necessário intimidade profunda com Deus e perceber os seus sinais em nosso tempo.

Jesus Ressuscitado, presente na Eucaristia, enche-nos de tua alegria e graça da ressurreição.

Neste Ano do Laicato, a Igreja estimula a presença e atuação dos cristãos leigos e leigas na vida eclesial e também na busca da transformação da sociedade. Vivamos intensamente este Ano Santo com as graças e o amor de Deus por nós!

Eu venho do Sul e do Norte, do Oeste e do Leste, de todo o lugar. / Estrada da vida eu percorro, levando socorro a quem precisar. / Assunto de paz é meu forte, eu cruzo montanhas e rou aprender. / O mundo não me satisfaz, o que eu quero é a paz, o que eu quero é viver.

No peito eu levo uma cruz, / no meu coração o que disse Jesus.

Momento da Palavra de Deus

Rezemos, para que nosso coração se abra ao amor de Deus para conosco e nos envie a graça de ouvir e seguir sua Palavra.

Meu Senhor e meu Pai, envia Teu Espírito Santo, para que eu compreenda e acolha a Tua Santa Palavra. Que eu te conheça e te faça conhecer, que eu te ame e te faça amar, que eu te sirva e te faça servir, te louve e te faça louvar, por todas as criaturas. Faze, ó Pai, que pela leitura de Tua Palavra, os pecadores se convertam, os justos perseverem na graça e todos consigamos a vida eterna. Amém!

É como a chuva que lava, é como o fogo que arrasa. / Tua palavra é assim: não passa por mim sem deixar um sinal. (×2)

1. Tenho medo de não responder, de fingir que eu não escutei. / Tenho medo de ouvir teu chamado, virar do outro lado e fingir que eu não sei. (×2)

2. Tenho medo de não perceber, de não ver teu amor passar. / Tenho medo de estar distraído, magoado, ferido e então me fechar. / Tenho medo de estar distraído, magoado, ferido e então me fechar.

O Senhor esteja conosco!

Ele está no meio de nós!

Proclamação do Evangelho de Jesus Cristo, segundo João.

FIGURE 9: A script of prayers for the Holy Hour.

and prayers arranged together, as well as several graphical elements throughout the document. The awesome tcolorbox package was used to generate the song boxes (represented by a music note icon in the upper right corner) and section styles, while the multicol package provided the layout with three balanced columns. For this particular project, we opted for narrower margins.

The music note icon in the upper right corner of a song box can be achieved by exploiting the overlay key in the tcolorbox environment. Having a vector image, we can use the following code:

```
\newcommand{\musicnote}{%
\begin{scope}[
  shift={
    ([xshift=-.5em,
      yshift=-.1em]frame.north east)
    }, scale=1 ]
   \node[ circle, fill=white,
          draw=black, inner sep=1pt
]{\includegraphics[scale=.6]{note}};
\end{scope}}
```

Figure 10 illustrates the final result, obtained by setting the `\musicnote` macro, previously defined, as overlay value in the tcolorbox environment corresponding to the song box.

Em nome do Pai, em nome do Filho, em nome do Espírito Santo, estamos aqui! (×2)

FIGURE 10: A note in the upper right corner of a song box.

Figure 11 presents a document containing a list of activities and events planned for the Holy Week

in 2019. We prepared it for distribution at the end of the masses during Lent. The schedule is typeset throughout two balanced columns and the activities of each day are grouped into boxes, powered by the tcolorbox package. Each activity also features an analog clock indicating the corresponding time (alongside the written counterpart, of course).

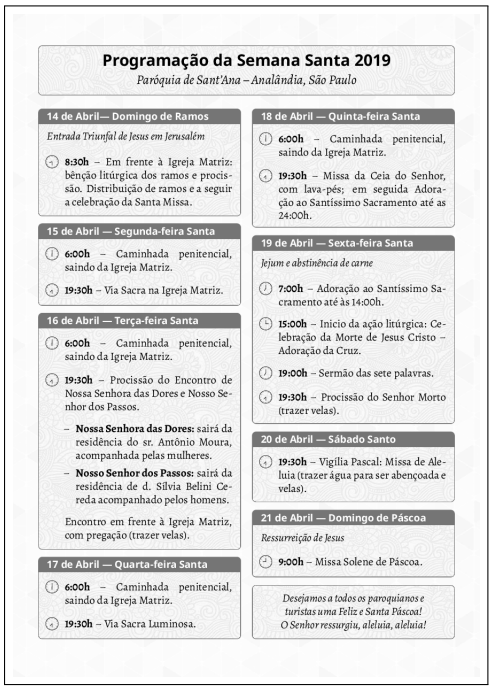


FIGURE 11: Activities and events for the Holy Week 2019.

I decided to include background images (simple patterns) for both page and each block as well, obtaining an interesting visual effect. In this case, I needed to ensure the opacity of such images did not compromise the text readability, so a couple of proofs were printed. This is a sample code of setting a background image in a tcolorbox environment:

```
underlay={\begin{tcbclipinterior}
  \node[anchor=south] at
     ([yshift=-10pt]interior.south)
  {\includegraphics[width=\tcb@width,
  keepaspectratio]{assets/bloom.png}};
\end{tcbclipinterior}}
```

The materials presented in this section do not contain any complex code at all. It is just a matter of proposing simple, minimalist layouts inspired by good, existing typographical works. Again, less is more, specially in a very specific domain such as church materials in which there is a higher, ultimate goal to achieve: contribute, through typesetting, to the complete fulfilment of a person's relationship with God.

4 A case study

On October 2020 in Assisi, at the tomb of Francis of Assisi in the basilica, after mass, Pope Francis signed *Fratelli tutti*, his third encyclical letter, calling for more human fraternity and solidarity, and is a plea to reject wars. The document is divided in 8 chapters, 287 paragraphs and contains 288 footnotes.

The complete text of this encyclical letter (as well as other encyclicals from Pope Francis and his predecessors) is freely available in the Vatican website for online browsing. Additionally, the Holy See provides a PDF version of such document. By inspecting the PDF metadata, it is quite possible to infer that the file is automatically generated.

As a case study, I humbly tried to propose a minimalist layout (Figure 12, second image) as an alternative to the official one (Figure 12, first image). Some remarks of my attempt:

1. The original document uses sans serif fonts. By inspecting it, the fonts used are Arial and Helvetica. I decided to rely on a serif font instead and use just one typeface throughout the entire document. For this particular experiment, I chose Karol, a proprietary font created in 2013 by Daniel Sabino. I used XeLaTeX as engine and included fontspec and microtype for proper font handling:

   ```
   \usepackage{fontspec}
   \usepackage{microtype}

   \setmainfont[Ligatures=TeX]{Karol}
   ```

2. I removed any colour from the document and went for a black and white approach. The Papal Insignia was desaturated as a means to have a monochrome version and match the proposed document style. I included graphicx for graphics support:

   ```
   \usepackage{graphicx}
   ```

3. I found the margins in the original document to be too narrow, so I made then wider. I also reduced the font size and kept the default line spacing and text alignment. I used memoir as document class with A4 as paper size:

   ```
   \documentclass[a4paper,
        12pt, article]{memoir}
   ```

4. Excerpts of text written in all capitals were rewritten to match sentence case, i.e, the standard case used in prose, in which only the first word is capitalized, except for proper nouns and specific words that are capitalized by certain rules. Only a few occurrences of sentences in all capitals were semantically replaced by small capitals as to emphasise key words:

   ```
   \scshape La Santa Sede
   ```

La Santa Sede

LETTERA ENCICLICA
FRATELLI TUTTI
DEL SANTO PADRE
FRANCESCO
SULLA FRATERNITÀ
E L'AMICIZIA SOCIALE

1. «*Fratelli tutti*»,[1] scriveva San Francesco d'Assisi per rivolgersi a tutti i fratelli e le sorelle e proporre loro una forma di vita dal sapore di Vangelo. Tra i suoi consigli voglio evidenziarne uno, nel quale invita a un amore che va al di là delle barriere della geografia e dello spazio. Qui egli dichiara beato colui che ama l'altro «quando fosse lontano da lui, quanto se fosse accanto a lui».[2] Con queste poche e semplici parole ha spiegato l'essenziale di una fraternità aperta, che permette di riconoscere, apprezzare e amare ogni persona al di là della vicinanza fisica, al di là del luogo del mondo dove è nata o dove abita.

La Santa Sede

Lettera Enciclica

Fratelli Tutti

del Santo Padre Francesco
sulla fraternità e l'amicizia sociale

§1 Fratelli Tutti[1], scriveva San Francesco d'Assisi per rivolgersi a tutti i fratelli e le sorelle e proporre loro una forma di vita dal sapore di Vangelo. Tra i suoi consigli voglio evidenziarne uno, nel quale invita a un amore che va al di là delle barriere della geografia e dello spazio. Qui egli dichiara beato colui che ama l'altro «quando fosse lontano da lui, quanto se fosse accanto a lui».[2] Con queste poche e semplici parole ha spiegato l'essenziale di una fraternità aperta, che permette di riconoscere, apprezzare e amare ogni persona al di là della vicinanza fisica, al di là del luogo del mondo dove è nata o dove abita.

Figure 12: The encyclical letter *Fratelli tutti* written by Pope Francis. The first image is the official document from the Holy See (available in the Vatican website), and the second image is my humble attempt for a minimalist layout.

Bold styles were completely removed whereas italics and emphasis were kept to a minimum.

5. The separator rule between the Papal Insignia and the encyclical letter title was made thinner, with the addition of a small ornament glyph in the centre (both vertical and horizontal), provided by the fourier-orns package:

```
\usepackage{fourier-orns}

\newcommand{\ornamentline}{{%
\parindent=0pt
\vspace{-1ex}\hrulefill%
\hspace{0.2cm} \raisebox{-1.5ex}
{\decoone} \hspace{0.2cm} \hrulefill%
}}
```

6. An initial was included in the first paragraph of the encyclical letter. An initial (sometimes referred as drop capital) is a letter at the beginning of a paragraph that is larger than the rest of the text and it usually spans two or more lines. I used Kramer Regular for the decorative initial font and the lettrine package for drop capital support.

```
% preamble
\usepackage{lettrine}

\input Kramer.fd
\newcommand*\initfamily{%
    \usefont{U}{Kramer}{xl}{n}}

% document body
\lettrine[nindent=2pt]{\initfamily F}
    {ratelli Tutti}\autocite{ref:1},
    scriveva...
```

7. Each paragraph of the encyclical letter is explicitly numbered, as a means to ease referencing and citing from other documents. Instead of using an enumerated list, I decided to keep the paragraphs as they are and include the corresponding number as a discrete marginal note. I prefixed it with a silcrow (i.e, a §) as to indicate the unique reference mark of the corresponding paragraph.

```
% Possible enhancement: set
% a counter and hook it on
% every paragraph

\newcommand{\parmark}[1]{%
\marginpar{\hspace{2em}\S#1} }
```

8. The original citation style was replaced to look like footnotes, so it would be not too intrusive. The bibliography is printed at the end of the document. I wonder how moving them to actual footnotes would actually look like. As I say, there is a lot of room for experimentation.

```
% Some suggestions...
\usepackage[style=verbose-ibid,
    backend=bibtex]{biblatex}
```

The final result is presented in the second image of Figure 12, alongside the original document. Of course, both versions convey the same message, as the texts are identical. However, I humbly advocate that the second version adds a layer of fine crafting that somehow might offer a subjective encouragement and motivation for potential readers.

5 Final remarks

There is an old and ubiquitous maxim in design that says that good typography is invisible. If I may add a complement to this sentence, I would say that good typography is also silent. Throughout the years, I noticed how a good design helps people focus on the actual message. Our efforts have to be in being good messengers, heralds of the content our design holds. Light has to go through our work. That is why I believe good typography is not just invisible, but silent as well: it is present in an apparent absence.

TEX plays an important role in my typographical adventures. The ecosystem is very rich and vibrant, offering countless opportunities to let the imagination and creativity soar. Typography is art. At the end of the day, there has to be a subjective component in which we can evaluate our efforts; good design has to be coherent and cohesive. Under this perspective, TEX and friends have to be at our services and not the other way around.

Good typography is indeed invisible and silent, but it is by no means imperceptible. Noticing it brings joy, gratification. Yet it is challenging to have this subtle perception. As once said in a Futurama episode, when you do things right, people will not be sure you have done anything at all.

References

Cereda, Paulo Roberto Massa (2020). «TEX in church: A typographical adventure». *TUGboat*, **42** (2), pp. 168–170.

▷ Paulo Roberto Massa Cereda
Analândia, São Paulo, Brazil
paulocereda at gmail dot com

Which Success for TEX as an Old Program?

Jean-Michel Hufflen

Abstract

We propose a personal analysis of TEX's strong and weak points. In particular, we show that this program's history can explain some conventions viewed as strange nowadays. From a point of view related to teaching LATEX to Computer Science students, going thoroughly into these strong and weak points allows us to show how this field has evolved over decades.

Sommario

In questo articolo è proposta un'analisi dei punti di forza e di debolezza di TEX. In particolare, si mostra come la storia di questo programma possa spiegare alcune convenzioni, che al giorno d'oggi possono sembrare poco familiari. Dal punto di vista di chi insegna LATEX a studenti che seguono corsi di informatica, esaminare scrupolosamente tali punti di forza e di debolezza consente di mostrare l'evoluzione, nel corso dei decenni, di questo campo di studi.

1 Introduction

The story has begun in 1978. Donald Knuth provided a powerful system, as a *kernel* and a *format*, *Plain TEX*, able to typeset texts. The adaptation of the notion of *document type*—originating from Scribe (Reid, 1984)—by Leslie Lamport resulted in a new format, LATEX, built out of TEX. LATEX is the most well-known format nowadays, especially within scientific publication, but also used in other topics, including history, humanities and social sciences. As another illustration of this success, the curricula of many universities and high schools include an introduction to LATEX. A good example of support material for such an introduction is given as part of last GUIr[1] meeting (2019). Of course, such an introduction is designed for PhD students, but sometimes also for graduate or undergraduate ones. This way, they can learn how to manage large-sized documents. Beyond LATEX's basic commands and environments, they can get aware of the separation of *data* and presentation, and more generally, the separation of *form* and *substance*.

TEX's kernel is a very old—and old-fashioned—program: as mentioned above, its first version came out in 1978. Current coding practices in programming have very little to do with the look of the languages of that time. Nevertheless, the typesetting systems built out of TEX are still widespread. Moreover, TEX's end has been announced many times... but the word processors based on TEX—LATEX, ConTEXt (Hagen, 2001), more recent—remain unrivalled for getting high-quality output prints. Only a few programs got a lifetime comparable to TEX's. But, due to some design choices, this program shows its age, even it is still at the forefront.

Hereafter we propose a personal analysis of strong and weak points of LATEX. We aim to emphasise that most of the weak points that perpetuate in recent versions result from design choices performed since the first versions. Teaching TEX & Co. to new end-users may be complicated because of these weak points, because they may be difficult to justify. But we think that Computer Science students—especially undergraduate ones—can get some experience from going thoroughly into this program. They can see how some initial design choices may become recurrent from a version to another. They can also discover how programming and software engineering have evolved since TEX's first version. In Section 2, we briefly recall the advantages of these typesetting systems. Section 3 is devoted to the some 'historical' conventions that may seem to be strange to new users. After discussing our way to introduce LATEX in Section 4, we conclude about our approach. Reading this farticle only requires basic knowledge about (LA)TEX, more precise details can be found in (Knuth, 1986a,b; Mittelbach *et al.*, 2004). Some technical points more related to Computer Science technique are explained in appendices: we tried to be precise for readers unfamiliar with Computer Science, without being too technical.

2 LATEX's advantages

As mentioned above, the typesetting systems built out of TEX are unrivalled for getting high-quality output prints. They are not interactive, they work like a programming language's compiler. This point may be viewed as a drawback in comparison with interactive WYSIWYG[2] systems. In fact, people writing very short reports or formatting documents that will be not reworked later may prefer interactive and graphical menus of systems such as Microsoft Word or InDesign. On the contrary, LATEX is often preferred by users writing large-sized doc-

1. ***G**ruppo **U**tilizzatori **I**taliani di TEX*.

2. '**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et', in contrast to WYSIWYM (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **M**ean) systems.

uments, or articles existing in different versions: a simple example can be given by an article included into a conference's proceedings, and later republished in a journal's issue, the layouts of these two versions may differ. For example, the former may be based on one-column layout, the latter on two-column layout.

TEX's kernel includes a kind of programming language, allowing users to develop new functions or customise existing ones. Roughly speaking, this language, mainly based on *macros*[3], is old-fashioned, and putting ambitious applications using it may be tedious, but many forums of LATEX users, organised by local user groups—e.g., GUIT—can help new programmers fix many mistakes. Anyway, such synergy among end-users is an important feature of *free software*. The same holds true for ConTEXt. LATEX's current version—LATEX 2ε—came out in 1994 (Lamport, 1994). As a suitable illustration of such synergy, this version provides many *packages* (Mittelbach *et al.*, 2004), developed by many users and whose fields are very diverse: some can be used for literary purposes, some are devoted to applications in Physics and Chemistry, etc. Many graphical tools usable around LATEX have been developed, too, as described in (Goossens *et al.*, 2009).

L. Lamport (1994) introduces LATEX 2ε as a higher-level system, in comparison with *Plain TEX*. This effort to move towards a very high-level programming language to express tasks LATEX can perform is in progress with the LATEX 3 project (Mittelbach and Schöpf, 1991). The result should be a language whose syntax is very different than LATEX 2ε's, this new language using precise notations for *types*, *variables*, *functions* and *evaluation strategies*. As shown in (LATEX3 project, 2020), the expl3 package allows end-users to experiment this new syntax. A recent survey is given in (Mittelbach and the LATEX3 project team, 2020).

Although TEX's end, as a legacy program's, was announced many times, new versions progressively incorporated modern requirements such as internationalisation or new schemes for font management. Moreover, the expressive power of engines based on TEX's kernel has been greatly increased by coupling it with a modern programming language. The most known example is LuaTEX (Hagen, 2006), where the engine can call procedures written using the Lua language (Ierusalimschy, 2006), other experiment connect TEX with Python (Lutz, 1996). Interesting applications based on such a *modus operandi* can be found in (Menke, 2019; Ziegenhagen, 2019). Developing such channels of communication between TEX and modern languages have succeeded, but let us mention that putting such extensions into action by using TEX's implementation language would have been perilous: on the one hand, this language is no longer practised, on the another hand, no one but D. Knuth is sufficiently familiar with TEX's sources in order to change them.

3. See Appendix A for more details about this notion.

3 LATEX's strange behaviour

If you practise LATEX for many years, you probably got used to these points for a long time. But if you have now to design an introduction to new end-users, some conventions will immediately appear as strange. Syntactic rules for mark-up are more homogeneous in LATEX than in *Plain TEX*[4]: a good example is given by the use of square brackets for commands' optional arguments. The recommended use of:

```
\begin{name} ... \end{name}
```

—where *name* is a command name dealing with sizes, such as `small` or `large`—rather than:

```
{ \name ... }
```

makes easier the use of such commands[5]. But let us remark that braces are often used to delimit arguments, and sometimes to limit local behaviour, what may be ambiguous. On the contrary, the duplicate commands for font style changes—e.g.:

```
\textbf{...} vs {\bfseries ...}
```

is related to the difference between *short* and *long* commands (Mittelbach *et al.*, 2004, Table 7.2): the latter can include an end-of-paragraph mark—that is the `\par` command—not the former. But this difference is less relevant than thirty years ago: at that time, running LATEX on a document that was about one hundred pages long took a while[6]. When LATEX processed such an end-of-paragraph mark, checking that there was no short command unclosed before continuing seemed more efficient than taking a while in order to reach the document's end, performing such check, and discovering that many commands were unclosed. But due to progress about computers' efficiency, this point is less justified nowadays.

TEX's language allows Computer Science students to deal with a *dynamic* language, whereas the programming languages used nowadays are *lexical*, in general (cf. Appendix B). When modern programming languages are used, compiling source files is based on both *lexical* and *syntactic* analyses. On the contrary, TEX's analyser of input streams has only one analyser, performing the whole of the process. As an illustration of old-fashioned

4. ... but such rules are more systematic in ConTEXt than in LATEX.

5. The same remark applies to LATEX's `sloppypar` environment, in comparison with the `\sloppy` command (Mittelbach *et al.*, 2004, p. 103).

6. We personally remembered that time, as reported in (Hufflen, 2003) in French.

syntactic conventions, a command's name must be followed by a separator, and a space character used as such is not put in the ouput built by LaTeX:

```
\LaTeX is beautiful
```
(1)

(it is well-known that a possible workaround is to replace '`\LaTeX ` ' by '`\LaTeX\ `'). Likewise, some strange behaviour result from the dynamic processing of conditional expressions: more examples are given in Appendix C.

In other situations, LaTeX aims to be ready for the next run. When we teach cross-references—by means of the commands `\label` and `\ref`—to students, they ask if it would possible to iterate LaTeX's running until solving cross-references reaches a stable state[7]. Here also, the answer is given by what happened many years ago: when a text was not finished, end-users may run LaTeX just in order to ensure that there was no syntactic mistake, but these end-users were not interested in checking an unfinished text's layout, they only aimed to know if they could go on later with their text. Let us recall that at that time, running LaTeX sometimes took a while...

According to a close point of view, LaTeX sometimes aims to be ready for the next run, provided that end-users help it. When LaTeX warns an end-user about an *overfull hbox*, often it allows this user to see where a word should be hyphenated. Even if a good technique is to fix that only on a text's final version, such convention yields good result, provided that end-users understand that LaTeX does not always build the best and final version, but allows users to approach it at next run. This point also holds on about marginal notes misplaced at a page's beginning. About vertical alignment of successive paragraphs on the same page, it has been said that TeX was either perfect, or deficient. In fact, if you aim to reach higher-level quality, these problems—orphans and widows—could be solved by using some commands such as `\looseness` (Knuth, 1986a, pp. 103–104). More recently, some advanced features of the microtype package (Schlicht, 2010)—*protrusion* and *font expansion*—based on the PDF[8] format—allow final texts to be reworked very precisely. This package also allows *tracking*[9]—that is, working on space between letters—by means of the commands `\textls` and `\SetTracking`, but this *modus operandi* should be very marginal. Anyway it should be reserved for very difficult cases, because it is not recommended within good typography, it should be applied to fragments using only capitals or only small capitals.

7. Let us remark that such iteration aiming to solve cross-references is done by ConTeXt's `texexec` command.

8. **P**ortable **D**ocument **F**ormat, Adobe's format.

9. This feature is also provided by the companion package letterspace, and by the soul package (Mittelbach *et al.*, 2004, § 3.1.7).

4 Discussion

As mentioned above, TeX's syntax is complicated, in comparison with LaTeX's. For example, LaTeX recommends an `\input` command's argument to be surrounded by braces—as any command's argument—whereas braces are not needed in *Plain TeX*. As any end-user may see, `$\frac12$` yields '$\frac{1}{2}$', that is '`1`' and '`2`' are obviously distinct tokens. That is not true for '`16`' in the following statement:

```
\write16{Do you enjoy \LaTeX?}
```

displaying the second argument at a terminal (Knuth, 1986a, pp. 226–228). In addition, braces are not allowed to surround an output stream number, the first argument of the `\write` command. LaTeX tries to deal with more systematic notations, but in general, what is usable in *Plain TeX* works within LaTeX. Some notations originating from *Plain TeX* are still used, because of bad habits. For example, '`$$...$$`' for a centered mathematical formula, although this convention is deprecated in LaTeX and should be replaced by '`\[...\]`' (Downes, 2017, § 2.1). *Plain TeX*'s conditional statements (cf. Appendix C) are old-fashioned and in LaTeX, we should use only the `\newif` command for simple cases (Knuth, 1986a, p. 211)) and the ifthen package (Mittelbach *et al.*, 2004, § A.3.2) or even better the etoolbox package (Lehman and Wright, 2020) for the others. But many source texts of packages use these old conditional statements. User-defined commands should be able to end with a space character if need be when the xspace pakage (Mittelbach *et al.*, 2004, § 3.1.2) is used. So, the behaviour pointed at (1) can be avoided. But this package sometimes makes a wrong decision, in which case some settings belonging to this package can be needed (Carlisle and Høgholm, 2014), putting '`{}`' at the end of such a command being another immediate solution.

We agree that some technical points may be avoided for non-Computer Science students. As an Assistant Professor of Computer Science, we think that Computer Science is... a *science*, and we have to teach this point to our students, who aim to be specialised in this topic. This science encompasses *techniques* and *methods*, but also has a *History*. Some programs got a very long lifetime, because they have been able to incorporate some new requirements, and a program such as TeX can illustrate that. Some programs are *monolithic* and it seems too difficult to rework the kernel: TeX belongs to this kind of programs, too. Some techniques have been experienced in Computer Science and have led to some behaviour difficult to understand and reproduce, or to some inconsistency: the problems raised by the way TeX parses its programs can show that. Some students have never

read any program using 'goto' statements: some examples chosen within (Knuth, 1986b) can illustrate this technique... and its drawbacks. This choice is relevant since TEX's implementation language, WEB[10], is quite structured, that is, the use of 'goto' statements is marginal, but instructive from a teaching point of view. Last, but not least, many students program as if they take no care of efficiency[11]. Here also, we can speak about NTS[12] (Taylor *et al.*, 2000). This rewriting of TEX in Java, developed at the 20th century's end, focused mainly on an object-oriented approach and neglected efficiency questions. As a result, this program worked much slower than the original TEX program and was unable to replace it.

To sum up, we think that the students who aim to become computer scientists can take advantage to the study of some obscure points of TEX. This program is still used: they can learn how it has been designed and how it works. This is an occasion of dealing with old techniques, not artificially. In fact, most of them enjoy that[13].

5 Conclusion

People who have to process short texts, or texts that will be not reworked, may be not interested in learning LATEX. The advantages of its approach are relevant for big-sized documents, possibly written by several authors. On another point, the curriculum of our students in Computer Science includes some *projects*, by groups. Every project ends with a report and an oral defence. We have noticed that many groups use LATEX for these two tasks, although they do not have to. Anyway, we can deduce that they enjoy using this typesetting program. The final touch of our lecture consists of a short introduction to Overleaf (2020), which is a collaborative cloud-based editor of LATEX documents and templates. Even if the LATEX version used may be not updated, even if no program can impose consistency about the style of several authors, students dealt with these aspects, and they noticed that LATEX was able to be adapted to such modern use. We began to experience this way last year, just before the Covid health crisis (!). Of course, we do not know how long it will last, but since teleworking is now encouraged, we think that our students follow the right path. To sum up, we are a happy teacher of LATEX, especially to Computer Science students. We hope that these students are happy, too, even if they go behind the scenes.

A Functions vs macros

Most often, *functions* belonging to modern programming languages use *calls by value*, that is, an argument is evaluated before applying this function. Let us consider the following *function*, written using Python, returning a list, its two elements being equal to the function's argument:

```
def twicef(x):
  return [x,x]
```

If you try to evaluate `twicef(2019 + 1)`, first the argument is evaluated to 2020, and then this value 2020 is bound to the local argument `x`. Of course, the result is the list `[2020,2020]`. Different techniques may be applied in other languages, including more modern techniques that optimise the evaluation of an argument—interested readers can find a survey in (Aho *et al.*, 2006)—but in general, an argument of such a subprogram is not evaluated several times[14]. Now let us consider this definition of a *macro* of the Scheme functional programming language[15]:

```
(define-syntax twice-m
  (syntax-rules ()
    ((twice-m x) ; Such a call is expanded to:
     (list (quote x) (quote x)))))
```

Let us recall that Scheme systematically uses prefixed operators, as shown by this evaluation:

```
(+ 2019 1) ⟹ 2020
```

but when a *macro* is called, every argument is processed *as it is*. Let us notice that the `quote` special form[16]—used within the expanded form of the `twice-m` macro—inhibits the evaluation of its argument, so:

```
(twice-m (+ 2019 1)) ⟹
        ((+ 2019 1) (+ 2019 1))
```

TEX's commands are closer to macros than functions[17]:

```
\def\twicem#1{[#1,#1]}
```

10. A good introduction to this language is (Knuth, 1992).

11. For example, ask them for a program checking if a string s is a palindrom. Many will answer $s = reverse(s)$, where *reverse* returns the reversed form of a string. Of course, such a solution would be acceptable for a specification, but not for a real program, that would perform more comparisons than needed.

12. **N**ew **T**ypesetting **S**ystem.

13. But we recognise that these obscure points of TEX have never been used within any examination!

14. ... except for some early programming languages... That was a long time ago...

15. A good introduction to this functional programming language, including its modern *hygienic* macro system, is (Dybvig, 1996).

16. Of course, `quote` is not a function.

17. By the way, let us mention that LATEX 3's terminology uses the *function* word (LATEX3 project, 2020), but according to a different sense in comparison with functional programming languages, even if programming *functions* according to this 'classical' sense is possible with expl3 (Gregorio, 2020). In fact, commands are divided into *variables* and *functions*, precise conventions rule available strategies for parameter passing.

That command builds its argument twice since it is put twice within the result. A simple test is given by an expression incrementing a counter and returning it[18]:

```
\newcounter{c}
\twicem{\stepcounter{c}\thec}
```

which yields '`[1,2]`'. If you would like this argument to be build only once, just put a *box* (Knuth, 1986a, pp. 120–122) as a container:

```
\newbox\tmpbox
\def\twicemversiontwo#1{%
 \setbox\tmpbox\hbox{#1}%
 [\unhcopy\tmpbox,\unhbox\tmpbox]}
```

and the expression:

```
 \twicemversiontwo{\stepcounter{c}\thec}
```

now yields '`[3,3]`'. Simpler cases can be solved by using local definitions or the `\expandafter` macro (Knuth, 1986a, p. 213).

B Lexically or dynamically

The difference between *lexical* and *dynamic* programming languages is related to *variables' scope*. Let us consider a subprogram that uses a notation not included into its arguments. When this subprogram is applied, a lexical (resp. dynamic) scope is put into action if we consider the value associated with this notation at definition-time (resp. runtime). Most programming languages used nowadays are lexical. About TEX, let us consider the following example, already given in (Hufflen, 2009):

```
\def\state{happy}
\edef\firstquestion{%
 You're \state, ain't U?\par}
\def\secondquestion{%
 You're \state, ain't U?\par}
\def\state{afraid}
```

In *Plain TEX*, many commands are introduced using the `\def` command (Knuth, 1986a, Ch. 20): such commands are dynamic, that is, other commands used inside bodies are expanded at runtime, so processing `\secondquestion` causes the paragraph:

You're afraid, ain't U?

to be typeset. By default, TEX is dynamic, but some lexical behaviour can be expressed using TEX's `\edef` command, because the whole body of such a command is fully expanded as soon as this command is defined. So, processing the `\firstquestion` command causes the paragraph:

You're happy, ain't U?

18. The `c` counter, defined hereafter, will be reused later.

```
def max3(x,y,z):
  if x >= y :
     if z >= x : return z
     else : return x
  elif z >= y : return z
  else : return y
```

Figure 1: Python program using conditional statements.

to be processed, even if the `\state` command has been redefined. LaTeX's kernel provides the `\protected@edef` command (Mittelbach *et al.*, 2004, p. 892), similar to `\edef`. However, LaTeX can be viewed as dynamic, since the preferred ways to define new commands—the `\newcommand` command and similar constructs mentioned in (Mittelbach *et al.*, 2004, § A.1.2)—introduce dynamic commands.

We have to pay particular attention when `\edef` is used for commands with arguments, since such a definition's body is expanded as far as possible *at definition-time*—some pitfalls are described in (Knuth, 1986a, pp. 215–216)—in particular, it is unsuitable to avoid an argument's multiple evaluation in the commands given in § A:

```
\def\twicemversionthree#1{%
 \edef\tmp{\noexpand#1}%
 [\tmp,\tmp]}
```

If the `\noexpand` command (Knuth, 1986a, p. 213) is removed within the previous definition, it crashes because the argument is unavailable at definition-time. Processing:

```
\twicemversionthree{\stepcounter{c}\thec}
```

yields '`[3,3]`', that is, the `c` counter has not been incremented, the argument is not evaluated.

C Processing conditional expressions

As an example of using conditional statements, Fig. 1 gives a Python function returning the greatest value among three numbers. In Python and most of modern programming languages, some constructs are specified by means of *reserved words*, e.g., '`if`', '`else`', '`elif`' (for 'else if'). Such a reserved word cannot be used as a variable and conventions based on *delimiters* allow us to use them as *literals*: '`'if'`', '`"else"`', ...

When such a source text is processed by an *interpreter* or *compiler*, two analyses—*lexical* and *syntactic*—are performed. The former aims to divide a source text into a sequence of *tokens*: e.g., '`if`', `x`', '`>=`', '`y`', ... The latter checks if the successive tokens are assembled in conformity with the programming language's grammar. The result of these two analyses may be viewed as a *tree*, Fig. 2

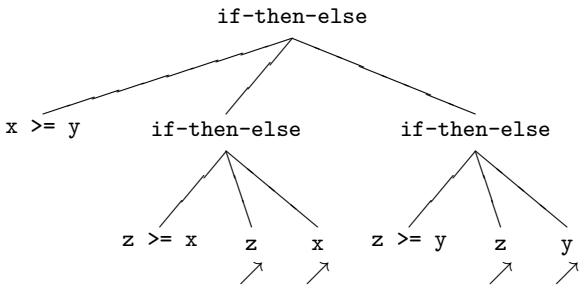


FIGURE 2: Abstract syntax tree for Fig. 1's program.

giving a sketch[19] of a tree modelling Fig. 1's program. So the *complete* structure of a source text is made explicit before interpreting this structure, or generating an executable program from it.

The *modus operandi* is very different within TEX. Only one analyser encompasses both lexical and syntactic analyses. More precisely, a command extracts its arguments as soon as it is recognised. Then TEX runs the command's body, and the process goes on. As a consequence, when TEX begins to process an '`\if...`' command, it does not know where this command will end. That is why we personally think that such process is fully *dynamic*, it may behave strangely if you get used to modern languages. Let us recall that all these '`\if...`' commands are documented in (KNUTH, 1986a, pp. 209–210): they end with a '`\fi`' marker. There may be an '`\else`' optional part, but we do not use this feature in the examples given below. They aim to emphasise the difference between recognising the beginning of an '`\if...`' command and popping a token, even if it could begin an '`\if...`' command.

(i) `\iffalse no\iftrue yes\fi more\fi`

In such a case, the `\iffalse` command causes the following tokens to be skipped until the `\fi` marker: as explained in (KNUTH, 1986b, §§ 500ff), skipping `\iftrue` adds a level and the two occurrences of `\fi` are correctly associated with `\iftrue` and `\iffalse`, respectively;

(ii) `\if\iftrue a\fi a ok\fi`

because of the `\if` command, the first token is expanded, the second, too, and 'ok' is typeset;

(iii) `\ifx\iftrue a ko\fi`

here, the `\ifx` command takes the first two tokens without expanding them, so the comparison just involves `\iftrue` and `a`, the `\fi` marker is associated to the `\ifx` command, which is quite counter-intuitive at first glance;

(iv) `\iftrue\verb+\iffalse+something\fi`

because of the `\iftrue` command, the following tokens will be processed until the `\fi` marker: `\iffalse` is processed as the argument of the `\verb` command and the `\fi` marker is associated with `\iftrue`;

(v) `\iffalse\verb+\iffalse+no\fi no\fi`

like in (i), the tokens following the first occurrence of `\iffalse` are skipped, but the second occurrence of `\iffalse` adds a level when it is skipped and the `\fi` marker's two occurrences are needed in order for this input line to be processed without crash, even if this behaviour is counter-intuitive[20].

Finally, let us remark that an '`\if...`' command marker can appear inside a body's command without being associated with a `\fi` marker and *vice-versa*[21]. A very good example is the '`\loop ... \repeat`' macro, provided by *Plain TEX* for expressing iterative operations (KNUTH, 1986a, pp. 217–218 & 352), where `\repeat` is defined this way:

```
\let\repeat=\fi
```

Acknowledgements

This work has been supported by the EIPHI Graduate school. Many thanks to Francesco Biccari for his patience and Massimiliano Dominici, who has translated the abstract in Italian. I am also very grateful to the reviewers, who addressed me constructive criticisms and suggested me some improvement points.

References

AHO, Alfred V., Monica S. LAM, Ravi SETHI and Jeffrey D.ULLMAN (2006). *Compilers, Principles, Techniques and Tools*. Pearson Internationel Edition, 2ª edizione.

CARLISLE, David and Moten HØGHOLM (2014). *The xpace package*. http://ctan.org/pkg/xspace.

DOWNES, Michael J. with Barbara BEETON (2017). *Short Math Guide for LATEX*. http://tug.ctan.org/info/short-math-guide.pdf.

DYBVIG, R. Kent (1996). *The Scheme Programming Language. ANSI Scheme*. Prentice-Hall, 2ª edizione.

19. An 'actual' abstract syntax tree would be more expanded: an expression such as '`x >= y`' would be replaced by a sub-tree, the same for '`...`↗', picturing a value to be returned.

20. Roughly speaking, the `\verb` command belongs to LATEX, not to TEX's kernel, so it may be told that mixing LATEX and *Plain TEX* is not recommended and may cause strange behaviour. This remark does not apply to Example (iii), which is counter-intuitive, too.

21. That is impossible within modern languages. In Python (cf. Fig. 1), an `if` construct ends with a line beginning at the left of the indentation point.

Goossens, Michel, Frank Mittelbach, Sebastian Rahtz, Denis B. Roegel and Herbert Voss (2009). *The LaTeX Graphics Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 2ª edizione.

Gregorio, Enrico (2020). «Funzioni e expl3». *ArsTEXnica*, **30**, pp. 36–45.

GuIT (a cura di) (2019). *Meeting, Sessione mattutina*. ArsTEXnica 28, pp. 8–134, Turin.

Hagen, Hans (2001). *ConTEXt, the Manual*. http://www.pragma-ade.com/general/manuals/cont-enp.pdf.

— (2006). «LuaTEX: Howling to the moon». *Biuletyn Polskiej Grupy Użytkowników Systemu TEX*, **23**, pp. 63–68.

Hufflen, Jean-Michel (2003). «Mes diverses périodes avec LaTeX». *Cahiers GUTenberg*, **42**, pp. 38–60.

— (2009). «Using TeX's language within a course about functional programming». *maps*, **39**, pp. 92–98. In EuroTEX 2009 conference.

Ierusalimschy, Roberto (2006). *Programming in Lua*. Lua.org, 2ª edizione.

Knuth, Donald Ervin (1986a). *Computers & Typesetting. Vol. A: The TEXbook*. Addison-Wesley Publishing Company, Reading, Massachusetts.

— (1986b). *Computers & Typesetting. Vol. B: The Program*. Addison-Wesley Publishing Company, Reading, Massachusetts.

— (1992). *How to read a WEB*, Center for the Study of Language and Information, capitolo 7, pp. 179–184. Numero 27 in Lecture Notes.

Lamport, Leslie (1994). *LaTeX: A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts.

LaTeX3 project (2020). *The LaTeX3 Interfaces*. http://ctn.org/pkg/l3kernel/interface3.pdf.

Lehman, Philipp and Joseph Wright (2020). *The etoolbox Package. An e-TEX Toolbox for Class and Package Authors*. ctan.org/pkg/etoolbox.

Lutz, Mark (1996). *Programming Python*. O'Reilly & Associates.

Menke, Henri (2019). «Parsing complex data formats in LuaTeX with lpeg». *TUGBoat*, **40** (2), pp. 129–135. In Proc. tug.

Mittelbach, Frank and Rainer Schöpf (1991). «Towards LaTeX 3.0». *TUGBoat*, **12** (1), pp. 74–79.

Mittelbach, Frank and the LaTeX3 project team (2020). «Quo vadis LaTeX(3) team—A look back and at the coming years». *TUGBoat*, **41** (2), pp. 201–207.

Mittelbach, Frank and Michel Goossens, with Johannes Braams, David Carlisle, Chris A. Rowley, Christine Detig and Joachim Schrod (2004). *The LaTeX Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 2ª edizione.

Overleaf (2020). *LaTeX, Evolved. The Easy to Use, Online, Collaborative Editor*. https://www.overleaf.com.

Reid, Brian Keith (1984). «scribe document production system user manual». Technical report, Unilogic, Ltd.

Schlicht, Robert (2010). *Microtype: an Interface to the Micro-Typographic Extensions of pdfTEX*. http://ctan.org/pkg/microtype.

Taylor, Philip, Jiři Zlatuška and Karel Skoupý (2000). «The NTS project: from conception to implementation». *Cahiers GUTenberg*, **35–36**, pp. 53–77.

Ziegenhagen, Uwe (2019). «Combining LaTeX with Python». *TUGBoat*, **40** (2), pp. 126–128. In Proc. tug 2019.

▷ Jean-Michel Hufflen
FEMTO-ST (UMR CNRS 6174) &
University of Bourgogne Franche-Comté,
16, route de Gray,
25030 Besançon CEDEX
France
jmhuffle at femto-st dot fr

europasscv: una classe non ufficiale per il curriculum vitae nel formato Europass

Giacomo Mazzamuto

Sommario

europasscv è una classe LaTeX per la composizione di un curriculum vitae nel formato Europass CV, il modello standard raccomandato dalla Commissione Europea. Il formato Europass CV, lanciato nel 2002, ha visto nel 2013 un aggiornamento dello stile che lo ha reso più moderno oltre che più pulito e compatto. La presente classe si riferisce proprio alla versione 2013 del formato Europass, del quale implementa soprattutto lo stile lasciando piena libertà all'utente su come strutturare i contenuti.

Abstract

europasscv is an unofficial LaTeX implementation of the Europass CV, the standard model for curriculum vitae as recommended by the European Commission. The style of the Europass CV, launched in 2002, was updated in 2013 with a major revision, featuring a neater, more compact and somewhat fancier layout. This class is an implementation of the 2013 version of that layout. While the Europass CV defines both the content and the layout of a curriculum vitae, the europasscv class mostly provides support for the latter, leaving the user full control on how to structure the content.

1 Introduzione

La classe europasscv permette di comporre un curriculum vitae nel formato "Europass CV", il modello standard raccomandato dalla Commissione Europea. Lanciato nel 2002, il modello è stato aggiornato nel 2013 con uno stile più moderno che è quello adottato in questa classe. A luglio 2020 è uscito un ulteriore aggiornamento, con ben tre stili differenti tra cui l'utente può scegliere. Ad oggi, Europass[1] è molto più di un modello grafico di curriculum vitae: si tratta di un'articolata piattaforma online che non solo permette di compilare il proprio CV tramite un'interfaccia molto semplice, ma offre svariati servizi come la possibilità di consultare offerte di lavoro e di studio e di inviare candidature, o la possibilità di creare un proprio profilo professionale consultabile dai datori di lavoro. Naturalmente la classe europasscv si occupa soltanto dell'aspetto puramente tipografico e non implementa alcune funzionalità che sono invece fruibili dal sito, come ad esempio l'inclusione all'interno del PDF dei dati relativi al CV in formato *machine-readable*[2].

Da quando è stato introdotto, il formato Europass ha riscosso un enorme successo. Ad ottobre 2017, tramite la piattaforma online erano stati generati 100 milioni di curriculum e scaricati 60 milioni di modelli[3]. A mio avviso, si tratta di un'iniziativa lodevole che fornisce un aiuto importante a chi si affaccia per la prima volta sul mondo del lavoro e magari non ha una chiara idea di come si scrive un CV valorizzando opportunamente la propria esperienza professionale e di studio. Sempre secondo la mia opinione, gli utenti più esperti dovrebbero però valutare attentamente l'opportunità di ricorrere a questo formato per evitare un'eccessiva standardizzazione che non differenzierebbe (nella forma) il proprio CV da tanti altri. Va detto però che, rispetto alla versione originale, le versioni successive della piattaforma online lasciano all'utente una certa libertà su come strutturare il proprio curriculum (nella versione 2020 anche per quanto riguarda lo stile). Tuttavia ci sono molti casi in cui una composizione pulita e ordinata come quella fornita da questa classe è tutto quello che serve (ad esempio per la partecipazione ad un concorso pubblico, come dimostrato dal discreto successo che questo pacchetto ha avuto anche in ambito accademico).

2 Guida all'uso

2.1 Uso generale

Per utilizzare questo pacchetto, occorre caricare la classe europasscv e creare l'omonimo ambiente:

```
\documentclass[italian,a4paper]{europasscv}
\begin{document}
  \begin{europasscv}
  \end{europasscv}
\end{document}
```

Per prima cosa, vanno specificate le informazioni anagrafiche quali: nome e cognome, indirizzo, informazioni di contatto, ecc. Ecco un esempio con i dati più comuni (si veda la documentazione per un elenco completo dei comandi):

1. https://europa.eu/europass/it

2. Nella fattispecie, il PDF generato online contiene un documento XML che permette di accedere programmaticamente alle informazioni in esso contenute.

3. https://www.cedefop.europa.eu/, sezione "News".

```
\ecvname{Name Surname}
\ecvaddress{rue Wiertz, B-1047 Brussels}
\ecvtelephone{(+555) 555 555}
\ecvmobile{(+555) 340 123}
\ecvemail{smith@kotmail.com}
\ecvhomepage{www.myhomepage.com}
\ecvdateofbirth{1 January 1970}
```

Si può anche inserire la propria fotografia con uno di questi comandi:

```
\ecvpictureleft[⟨options⟩]{filename}
\ecvpictureright[⟨options⟩]{filename}
```

dove le opzioni sono passate direttamente al sottostante comando `\includegraphics`. Fino a qui si tratta di comandi di definizione, che non comportano ancora nessun tipo di output. Per comporre sul documento il blocco contenente le informazioni personali, ciascuna affiancata dalla corrispondente icona, si utilizza quindi il comando:

```
\ecvpersonalinfo
```

Dopo le informazioni personali, si passa al contenuto vero e proprio del curriculum. Il formato Europass, nella versione in questione, ha un *layout* diviso verticalmente in due parti. A sinistra, una colonna pari in larghezza a circa un quarto della pagina viene usata per specificare informazioni aggiuntive che fungano da "guida per l'occhio" nello scorrere il curriculum (ad esempio: data di inizio e fine dell'attività lavorativa, titoli delle sottosezioni, ecc.). A destra, il rimanente spazio in larghezza viene utilizzato per il corpo del curriculum dove sono descritte le varie attività. Per aggiungere un elemento si usa il comando seguente, dove per ogni riga inserita si può specificare il testo per la colonna di sinistra e quello per la colonna di destra:

```
\ecvitem[⟨vspace⟩]{left}{right}
```

L'argomento opzionale `vspace` consente di specificare lo spazio verticale da lasciare vuoto *prima* dell'elemento appena aggiunto. Dato che la colonna di sinistra è più stretta di quella di destra, l'argomento `left` sarà una stringa preferibilmente corta (molto spesso sarà addirittura una stringa vuota). Ad esempio si può usare questo argomento per specificare l'intervallo di date a cui quell'elemento si riferisce. Per sfruttare al meglio questo *layout* è importante non esagerare con il contenuto posto nella colonna di sinistra, è anzi opportuno dosarne l'uso al fine di creare un effetto di sezionamento ad una visione d'insieme del CV. L'argomento `right` potrà invece contenere anche interi paragrafi o elenchi, che si possono ottenere con l'ambiente `ecvitemize`.

Ci sono molte varianti del comando `\ecvitem` che consentono di inserire elementi con stili diversi tra cui:

```
\ecvtitle[⟨vspace⟩]{left}{right}
\ecvblueitem[⟨vspace⟩]{left}{right}
\ecvbigitem[⟨vspace⟩]{left}{right}
```

Per creare delle sezioni con titolo, il comando da usare è il seguente:

```
\ecvsection[⟨vspace⟩]{title}
```

La figura 1 mostra una carrellata di tutti i comandi di sezionamento disponibili e di qualche comando di utilità per evidenziare testo o intere celle; in figura 2 è riportato il corrispondente codice.

La seconda parte del curriculum è solitamente dedicata alle competenze personali, le quali possono essere inserite usando i soliti comandi di sezionamento descritti sopra. Per quanto riguarda la padronanza delle lingue straniere, il formato Europass prevede una tabella di autovalutazione nella quale si può indicare il proprio livello di comprensione scritta e parlata e di produzione orale. Nella classe `europasscv` questa tabella può essere inserita con i seguenti comandi:

```
\ecvmothertongue{English}
\ecvlanguageheader
\ecvlanguage{French}{C1}{C2}{B2}{C1}{C2}
\ecvlanguagecertificate{Diplôme d'études en
    langue française (DELF) B1}
\ecvlastlanguage{German}{A2}{A2}{A2}{A2}{A2}
\ecvlanguagefooter
```

Un esempio di come la tabella viene resa nel documento è mostrato in figura 3. Infine, è prevista una tabella analoga per le competenze digitali.

2.2 Funzionalità avanzate

La classe `europasscv` supporta diverse opzioni, ad esempio per cambiare la lingua o il font, o per nascondere il logo Europass. Per un elenco completo e per le istruzioni dettagliate si rimanda alla documentazione[4].

Una funzionalità particolarmente interessante in ambito accademico, che vale qui la pena di illustrare, riguarda l'inclusione della propria bibliografia all'interno del CV, intesa come l'elenco delle opere pubblicate a proprio nome. In particolare, si può automaticamente evidenziare in grassetto il proprio nome all'interno dell'elenco degli autori di ogni elemento bibliografico. Per attivare questa funzionalità, è sufficiente il seguente comando:

```
\ecvbibhighlight{last name}
                {first name}
                {first name initials}
```

L'inclusione della bibliografia nel CV richiede comunque di utilizzare un pacchetto specifico (`europasscv-bibliography`) e alcuni comandi appositi per i quali si rimanda alla documentazione.

4. https://ctan.org/pkg/europasscv

JOB APPLIED FOR ecvbigitem

ECVSECTION

apr 2012 – apr 2014 **ecvtitle**

ecvitem: this is the standard CV item. This is some highlighted text.

ecvitemize
- this is a list item
- this is a list item
  - this is a list item
  - this is a list item

blue left text ecvblueitem

apr 2012 – apr 2014 **ecvtitlelevel** ISCED 6

ecvitem Item description

ecvitem highlighted cell

FIGURA 1: Panoramica dei comandi di sezionamento.

```
\ecvbigitem{Job applied for}{ecvbigitem}

\ecvsection{ecvsection}
\ecvtitle{apr 2012 -- apr 2014}{ecvtitle}
\ecvitem{}{ecvitem: this is the standard CV item.
           This is some \ecvhighlight{highlighted} text.}
\ecvitem{ecvitemize}{
    \begin{ecvitemize}
        \item this is a list item
        \item this is a list item
        \begin{ecvitemize}
            \item this is a list item
            \item this is a list item
        \end{ecvitemize}
\end{ecvitemize}
}
\ecvblueitem{blue left text}{ecvblueitem}
\ecvtitlelevel{apr 2012 -- apr 2014}{ecvtitlelevel}{ISCED 6}
\ecvitem{ecvitem}{Item description}
\ecvitem{ecvitem}{\ecvhighlightcell{highlighted cell}}
```

FIGURA 2: Codice sorgente per la figura 1.

Lingua madre Italiano

Altre lingue	COMPRENSIONE		PARLATO		PRODUZIONE SCRITTA
	Ascolto	Lettura	Interazione	Produzione orale	
Francese	B1	B1	B1	B1	B1
	Diplôme d'études en langue française (DELF) B1				
Inglese	C1	C2	B2	C1	C2

Livelli: A1 e A2: Utente base – B1 e B2: Utente autonomo – C1 e C2: Utente avanzato
Quadro Comune Europeo di Riferimento delle Lingue

FIGURA 3: Esempio di tabella di autovalutazione delle competenze linguistiche.

europass Curriculum vitae

PERSONAL INFORMATION **Katie Smith**

12 Strawberry Hill, Dublin 8 Éire/Ireland

+353 555 123 555 ☎ +353 127 6689 +353 999 888 777

smith@kotmail.com another@email.com

www.myhomepage.com www.another-homepage.com

AOL Messenger katie.smith Google Talk ksmith

Date of birth 1 March 1975 | Nationality Irish

JOB APPLIED FOR European project manager

WORK EXPERIENCE

August 2002 – Present **Independent consultant**

National Youth Council of Ireland
3 Montague Street, Dublin 2, D02 V327, Ireland

Evaluation of European Commission youth training support measures for youth national agencies and young people

March 2002 – July 2002 **Internship**

European Commission, Youth Unit, DG Education and Culture
200, Rue de la Loi, 1049 Brussels (Belgium)

- evaluating youth training programmes and the partnership between the Council of Europe and European Commission
- organizing and running a 2 day workshop on non-formal education for Action 5 large scale projects focusing on quality, assessment and recognition
- contributing to the steering sroup on training and developing action plans on training for the next 3 years. Working on the Users Guide for training and the support measures

Business or sector European institution

Oct 2001 – Feb 2002 **Researcher / Independent Consultant**

Council of Europe, Budapest (Hungary)

Working in a research team carrying out in-depth qualitative evaluation of the 2 year Advanced Training of Trainers in Europe using participant observations, in-depth interviews and focus groups. Work carried out in training courses in Strasbourg, Slovenia and Budapest.

EDUCATION AND TRAINING

1997–2001 **PhD - Thesis Title: "Young People in the Construction of the Virtual University", Empirical research on e-learning** ISCED 6

Trinity College Dublin, The University of Dublin, Ireland

This is a citation to my PhD Thesis [1].

1993–1997 **Bachelor of Science in Sociology and Psychology**

Trinity College Dublin, The University of Dublin, Ireland

- sociology of risk
- sociology of scientific knowledge / information society
- anthropology
- E-learning and Psychology
- research methods

PERSONAL SKILLS

Mother tongue English

Other languages

	UNDERSTANDING		SPEAKING		WRITING
	Listening	Reading	Spoken interaction	Spoken production	
French	C1	C2	B2	C1	C2
	Diplôme d'études en langue française (DELF) B1				
German	A2	A2	A2	A2	A2

Levels: A1 and A2: Basic user – B1 and B2: Independent user – C1 and C2: Proficient user
Common European Framework of Reference for Languages

Communication skills

- team work: I have worked in various types of teams from research teams to national league hockey. For 2 years I coached my university hockey team
- mediating skills: I work on the borders between young people, youth trainers, youth policy and researchers, for example running a 3 day workshop at CoE Symposium "Youth Actor of Social Change", and my continued work on youth training programmes
- intercultural skills: I am experienced at working in a European dimension such as being a rapporteur at the CoE Budapest "youth against violence seminar" and working with refugees.

Organisational / managerial skills

- whilst working for a Brussels based refugee NGO "Convivial" I organized a "Civil Dialogue" between refugees and civil servants at the European Commission 20th June 2002
- during my PhD I organised a seminar series on research methods

Digital competences

SELF-ASSESSMENT				
Information Processing	Communication	Content creation	Safety	Problem solving
Basic user	Independent user	Proficient user	Independent user	Basic user

Digital competences - Self-assessment grid

Computer skills

- competent with most Microsoft Office programmes
- experience with HTML

Other skills Creating pieces of Art and visiting Modern Art galleries. Enjoy all sports particularly hockey, football and running. Love to travel and experience different cultures.

Driving licence A, B

PUBLICATIONS

[1] **Katie Middlename Smith Testington**. "Young People in the Construction of the Virtual University. Empirical research on e-learning." PhD thesis. Dublin, 2001.

[2] **K. M. Smith Testington** and B. Boss. "Automatic use of a bold font in the publication list to highlight a particular author." In: *International Journal of Advanced LaTeX usage* (2015).

[3] **Katie Middlename Smith Testington**. "How to do Observations: Borrowing techniques from the Social Sciences to help Participants do Observations in Simulation Exercises". In: *Coyote EU/CoE Partnership Publication* (2002).

FIGURA 4: Esempio di curriculum vitae composto con la classe `europasscv`.

2.3 Lettera di accompagnamento

A partire dall'ultima versione della classe `europasscv` è stata aggiunta la possibilità, da tempo richiesta, di comporre una *cover letter* che faccia da accompagnamento alla presentazione del CV. All'interno dell'ambiente `europasscv` si può inserire una cover letter con questi semplici comandi:

```
\thispagestyle{empty}
\eclpersonalinfo

\ecladdressee{Contact person}
             {Name of organization}
             {Address of organization}
             {City}

\eclcitydatesubject{City}
                   {\today}
                   {Subject of cover letter}

\eclopeningsalutation{Dear Sir}
\eclmaincontent
{Opening salutation.\bigskip}

{This is the main content.}

{Closing salutation.}

\eclclosingsalutation{Yours sincerely}

\eclsignature
%\eclsignature[signature.jpg]
%\eclsignature[signature.jpg][Dr.\ JK Smith]

\pagebreak \setcounter{page}{1}
```

3 Sviluppo e prospettive future

Come spesso accade nel mondo *open source*, lo sviluppo di questo pacchetto è partito da un'esigenza personale. Su CTAN esisteva già il pacchetto `europecv`, sviluppato da Nicola Vitacolonna, che si riferiva però alla prima versione del curriculum europeo. Lo sviluppo è quindi partito proprio dalla classe `europecv`, che è stata pesantemente modificata nel corso di 10 versioni di `europasscv` ad oggi rilasciate a partire da marzo 2015. La parte relativa alla lettera di accompagnamento è stata invece presa, con opportune modifiche, dalla classe `europecv2013`[5] di Roberto Leinardi. Entrambe queste classi sembrano non essere più in uno stato di sviluppo attivo da alcuni anni.

Nonostante le profonde modifiche e l'aggiunta di molti comandi, la sintassi di `europasscv` e l'organizzazione interna della classe (spesso non molto elegante, sia nelle vecchie sia nelle nuove funzionalità), derivano ancora da quelle della classe originale. L'idea era quella di assicurare una certa retrocompatibilità con la classe originale, in modo che gli utenti potessero aggiornare il proprio CV semplicemente cambiando la classe e ricompilando. Sicuramente c'è molto spazio per migliorare l'organizzazione del codice e la resa del documento, ma forse vale la pena ricominciare da capo visti i tre nuovi *layout* lanciati a luglio 2020.

Con la versione 2020 di `europasscv` si chiude quindi un ciclo, in quanto a partire da quest'anno il modello di CV prodotto da questa classe non è quello più aggiornato. Lo sviluppo comunque non si ferma. Per chiunque volesse contribuire, lo sviluppo si svolge su GitHub[6], la nota piattaforma di *hosting* di codice, anche se forse non così popolare nel mondo LaTeX come lo è in altri ambiti di sviluppo.

▷ Giacomo Mazzamuto
Consiglio Nazionale delle Ricerche - Istituto Nazionale di Ottica (CNR-INO)
`g dot mazzamuto+ctan at gmail dot com`

5. https://github.com/leinardi/europecv2013
6. https://github.com/gmazzamuto/europasscv/

Funzioni e expl3

Enrico Gregorio

Sommario

Esporremo alcune nozioni relative alle funzioni nel linguaggio expl3: il loro ruolo, come si definiscono, le varianti; per le applicazioni, si darà anche un cenno alle variabili.

Abstract

In this tutorial we discuss expl3 functions, their role, definition and variants, touching also variables.[1]

1 Introduzione

Il termine *funzione* non è usato nel modo di esprimersi usuale in LaTeX. Invece è un concetto molto importante nel linguaggio expl3, che sarà alla base del futuro LaTeX3. Il linguaggio non aveva un nome preciso fino a un paio d'anni fa, quando fu deciso di chiamarlo expl3 come il pacchetto che ne permette l'uso. Nelle versioni attuali di LaTeX il pacchetto è stato incorporato nel nucleo.

In questo linguaggio, si ha cura di distinguere fra *variabili* e *funzioni*. Alle variabili si dà un valore che può cambiare durante la compilazione, mentre le funzioni eseguono una certa azione.

Un tipo speciale di variabili sono le *costanti*, il cui valore non deve cambiare durante la compilazione di un documento. Vero, il nome sembra contraddittorio, ma i matematici sono abituati a questo tipo di estensione della terminologia: chi non è matematico, non si preoccupi e vada avanti, limitandosi a sorridere delle bizzarrie del modo di pensare dei matematici.

Saremo principalmente interessati alle funzioni; tuttavia le variabili possono essere quello che passiamo alle funzioni, quindi ci servirà qualche nozione anche su queste.

Facciamo un esempio con concetti 'tradizionali'. Le classi di documento standard, e anche molte altre, possiedono il *comando* `\title`. Come funziona? Questo stesso articolo ha

```
\title{Funzioni e \expliii}
```

all'inizio. Quando LaTeX elabora questa istruzione, eseguirà qualcosa come

```
\gdef\@title{Funzioni e \expliii}
```

così da poter usare `\@title` durante la successiva elaborazione di `\maketitle`.

C'è una grande differenza concettuale tra `\title` e `\@title`. Il primo esegue un'azione, il secondo è semplicemente un contenitore. Nella terminologia di expl3, il primo sarà una funzione, il secondo una variabile: l'azione della funzione è di registrare qualcosa nella variabile specificata.

Al livello dell'utente, la distinzione non è così netta. Se scrivo

```
\newcommand{\CC}{\mathbb{C}}
```

sto dicendo che `\CC` è una funzione o una variabile? Non è veramente importante deciderlo, perché è piuttosto un'*abbreviazione* dell'utente e a questo livello la distinzione è quasi irrilevante.

La distinzione diventa rilevante quando si *programma*. Nella programmazione expl3 'corretta', ci sarà un comando a livello utente che chiama una funzione, la quale eseguirà un'opportuna azione:

```
\NewDocumentCommand{\title}{m}
 {
  \example_title:n { #1 }
 }
\tl_new:N \g_example_title_tl
\cs_new_protected:Nn \example_title:n
 {
  \tl_gset:Nn \g_example_title_tl { #1 }
 }
```

In seguito, il comando a livello utente `\maketitle` farà uso del valore registrato nella variabile, adoperando certamente altre funzioni. Nel seguito, *comando* si riferirà a ciò che si vede a livello utente, come `\CC` di prima.

Questo esempio (immaginario) mostra un buon numero dei concetti che discuteremo. Definiamo un comando utente in termini di una funzione; questa funzione ha un argomento (il titolo) e il suo mestiere è di impostare una ben determinata variabile con il valore specificato. La variabile va dichiarata prima di essere usata, tipicamente prima che si definisca una funzione che la impieghi:

- `\title` è un comando per l'utente, fuori dallo scopo di questo articolo;
- `\example_title:n` è una funzione;
- `\g_example_title_tl` è una variabile.

2 Convenzioni sui nomi

Un problema comune con TeX è che non possiede il concetto di *namespace*, che si diffuse nell'informatica parecchio più tardi. Conflitti di nomi erano

1. This article is a translation of a forthcoming paper in TUGboat.

molto frequenti ai primi tempi di LaTeX e continuano ad apparire di tanto in tanto. Potrebbe essere accattivante per un pacchetto adoperare `\@x` e `\@y` come coordinates, tuttavia gli autori del pacchetto dovrebbero rendersi conto che se gli piace un nome semplice è probabile che sia già piaciuto a qualcun altro.

Le variabili devono avere un nome della forma

`\l_`⟨*prefisso*⟩`_`⟨*nome proprio*⟩`_`⟨*tipo*⟩
`\g_`⟨*prefisso*⟩`_`⟨*nome proprio*⟩`_`⟨*tipo*⟩
`\c_`⟨*prefisso*⟩`_`⟨*nome proprio*⟩`_`⟨*tipo*⟩

dove le varie parti sono *importanti* e *necessarie*:

- `l`, `g` o `c` dichiarano che la variabile that the variable è locale, globale o costante rispettivamente;
- ⟨*prefisso*⟩ dev'essere una stringa di lettere che identifichi univocamente il pacchetto o il codice nel documento;
- ⟨*nome proprio*⟩ è un'arbitraria stringa di lettere che è possibile dividere in parti separate da un *underscore*;
- ⟨*tipo*⟩ è il tipo di variabile.

I tipi di variabili più comuni sono:[2]

- `tl`, per *token list*;
- `seq`, per *sequence*;
- `clist`, per *comma list*;
- `prop`, per *property list*;
- `int`, per *integer*;
- `dim`, per *dimension*;
- `box`, per *box*;
- `fp`, per *floating point.*

Ce ne sono parecchie altre, ma lo scopo di quest'introduzione è discutere le funzioni, quindi non menzionerò le variabili più esoteriche, se non ce ne sarà bisogno.

I nomi delle funzioni sono simili:

`\`⟨*prefisso*⟩`_`⟨*nome proprio*⟩`:`⟨*segnatura*⟩

Il ⟨*prefisso*⟩ e il ⟨*nome proprio*⟩ sono come prima; la ⟨*segnatura*⟩ deve essere spiegata. Può essere un'arbitraria stringa di caratteri tra

`N n e f x T F c V v o w`

ciascuno dei quali, eccetto `w`, denota un argomento della funzione.

Le funzioni matematiche possono dipendere da uno o più argomenti (anche zero, quando sono funzioni costanti) e lo stesso vale per le funzioni di expl3. Lo scopo della segnatura è di dire precisamente da quanti argomenti dipende una funzione e anche il loro tipo. Per esempio, la funzione di uso frequente

```
\seq_set_split:Nnn
```

ha tre argomenti, uno di tipo `N` e due di tipo `n` *in quest'ordine*. Un argomento di tipo `N` dev'essere un solo token, la cui natura dipende ovviamente dalla funzione; nel caso in esame, dev'essere una variabile *sequence*. Un argomento di tipo `n` dev'essere una lista di token tra graffe. Per esempio, il titolo del documento è un argomento di tipo `n` a `\title`. Esempio un po' tirato per i capelli, ma dovrebbe spiegare il concetto.

Il tipo `w` è un'eccezione perché non dice quasi nulla, ma solo che gli argomenti della funzione sono *strani*, in inglese *weird*, e ci si deve riferire alla documentazione del pacchetto per sapere come regolarsi. In generale, argomenti di tipo `w` dovrebbero comparire solo in funzioni di basso livello.

Una chiamata della funzione menzionata prima sarà qualcosa come

```
\seq_set_split:Nnn
  \l_example_test_seq
  { || }
  { a || b || c }
```

(quasi certamente su una sola riga, qui è divisa per via del formato tipografico). Non ci interessa veramente sapere che cosa fa questo codice: questa funzione normalmente viene chiamata nel corso di altre elaborazioni e riceve gli argomenti da altre funzioni. L'aspetto importante è vedere che gli argomenti seguono le convenzioni: il primo è un singolo token, gli altri due liste di token tra graffe.

Una funzione può anche non avere argomenti, ma i due punti della segnatura sono necessari comunque. Per la verità, TeX potrebbe non protestare se definite una funzione con un nome non conforme alle convenzioni; seguirle aiuta a evitare conflitti e ad avere codice più leggibile. Una tipica funzione senza argomenti è `\scan_stop:`, nient'altro che il nostro vecchio amico `\relax`. Il nome in expl3 è forse meno poetico, ma esprime bene qual è lo scopo principale della funzione.

3 Come definire funzioni

Ci sono molte funzioni il cui mestiere è definire funzioni. Tutte, però, hanno lo stesso prefisso `cs` e le principali sono

```
\cs_new:Nn
\cs_new_protected:Nn
```

2. Non traduco i nomi perché avrebbe poco senso.

Discuteremo poi le altre. Secondo le convenzioni, queste due hanno due argomenti: il primo è un singolo token, cioè il nome della funzione da definire, il secondo è il *testo di sostituzione* (replacement text), cioè il codice che sarà sostituito quando la funzione è chiamata.

Sebbene expl3 cerchi di emulare un linguaggio funzionale, è pur sempre basato su TEX che conosce solo primitive, macro e registri. Questo dovrebbe essere sempre tenuto presente quando si programma. D'altra parte, il linguaggio permette costruzioni più semplici che evitano le goffe (o divertenti, a seconda dello spirito del programmatore) catene di `\expandafter` o `\noexpand`, che sono spesso difficili da seguire.

Un esempio semplice potrebbe essere la funzione interna per manipolare il titolo del documento:

```
\cs_new_protected:Nn \example_title:n
 {
  \tl_gset:Nn \g_example_title_tl { #1 }
 }
```

Siccome la funzione ha segnatura `:n` (per precisione, i due punti non sono parte della segnatura, ma sono convenienti come marcatore), expl3 sa che il testo di sostituzione può contenere `#1` per riferirsi all'argomento specificato alla chiamata della funzione.

Perché adopero l'istruzione con `protected` e non quella più semplice? Perché la nostra funzione deve *impostare* il valore di una variabile. Si tratta di un aspetto che ha frustrato generazioni di programmatori LATEX e ha richiesto la distinzione tra comandi robusti e fragili. Al giorno d'oggi il problema è meno rilevante perché quasi tutti i comandi fragili sono stati 'resi robusti', ma si può ancora presentare.

Qual è il problema? Se, in LATEX tradizionale, facciamo qualcosa come

```
\newcommand{\foo}[1]{%
  \renewcommand{\baz}{#1}%
}
```

che è il modo normale di immagazzinare qualcosa in un contenitore e per qualche motivo `\foo` finisce in `\write` o `\edef`, anche nelle forme `\protected@write` o `\protected@edef`, apparirà una lunga lista di messaggi di errore. Il modo tradizionale di evitare la faccenda è di adoperare `\DeclareRobustCommand` invece di `\newcommand`.

Ogni funzione il cui mestiere comprenda impostare variabili o chiamare altre funzioni 'protette' deve, in generale essere essa stessa 'protetta'. In caso di dubbio, si adotti la protezione.

Attenzione! La segnatura della funzione da definire può solo contenere i caratteri `N` o `n`. Per la verità, anche `T` o `F` sarebbero permessi, ma questo è un caso speciale che toccheremo più avanti. Come entrano allora gli altri caratteri ammessi nelle segnature? Ottima domanda!

3.1 Generare varianti

Supponiamo di dover scrivere una funzione generica per impostare variabili `tl` in modo che contengano materiale da usare in seguito. L'interfaccia utente avrà i comandi `\setvar` per impostare la variabile e `\usevar` per ottenerne il valore.

Abbiamo un dilemma: come fa l'utente a specificare il nome di una variabile all'interno del documento, dove i nomi expl3 non sono permessi? Infatti, nel testo normale l'underscore non può essere impiegato nel nome di un comando e quindi

```
\setvar{\l_example_var_a_tl}{something}
```

avrebbe effetti disastrosi. Preferiremmo, invece, che l'utente scriva

```
\setvar{a}{something}
\usevar{a}
```

(in punti diversi del documento, chiaro). Vediamo, a passo lento, come si può fare; l'interfaccia utente sarà sistemata dopo. Per prima cosa definiamo una funzione che allochi una variabile e la imposti a un valore; poi la funzione che fornisce il contenuto di una variabile:

```
\cs_new_protected:Nn \example_setvar:Nn
 {
  \tl_clear_new:N #1
  \tl_set:Nn #1 { #2 }
 }
\cs_new:Nn \example_usevar:N
 {
  \tl_use:N #1
 }
```

La prima istruzione cancella il valore della variabile specificata, se esistente, oppure ne alloca una nuova. La seconda funzione non deve essere protetta, perché semplicemente produce il valore e non fa nulla di pericoloso. Ma questo non risolve il dilemma. Ecco dove il concetto di *variante* entra in scena. Ora eseguiamo

```
\cs_generate_variant:Nn
  \example_setvar:Nn
  { cn }
\cs_generate_variant:Nn
  \example_usevar:N
  { c }
```

che definisce due *nuove* funzioni di nome

```
\example_setvar:cn
\example_usevar:c
```

Che significa `c`? Che la nuova funzione si aspetta un argomento tra graffe, dal quale costruirà un nome di comando (in questo caso sarà il nome di una variabile, in altri casi potrebbe essere il nome di una funzione). Ora possiamo definire l'interfaccia utente:

```
\NewDocumentCommand{\setvar}{mm}
 {
  \example_setvar:cn
    { l_example_var_#1_tl }
    { #2 }
 }
\NewExpandableDocumentCommand{\usevar}{m}
 {
  \example_usevar:c
    { l_example_var_#1_tl }
 }
```

I siti su LaTeX sono infestati da domande su codice che fa stranezze come `\def\c{something}` e che, chissà perché, produce errori. Con l'approccio appena delineato, stiamo effettivamente definendo un *namespace* per le nostre variabili, che quindi possono essere chiamate con il loro 'nome esterno', lasciando all'implementazione i dettagli su come evitare conflitti.

Potremmo naturalmente definire i comandi a livello utente in LaTeX tradizionale. Una tipica implementazione sarebbe

```
\newcommand{\setvar}[2]{%
  \expandafter
  \def\csname example@var@#1\endcsname{#2}%
}
\newcommand{\usevar}[1]{%
  \csname example@var@#1\endcsname
}
```

Non litigherò con chi insista che questo modo è più semplice. Ma rimarrò della mia opinione che non lo è.

Mettiamo qualcos'altro sul fuoco. Vogliamo permettere all'utente di copiare il contenuto di una variabile in un'altra con `\copyvar{b}{a}`, dove `b` è la nuova 'variabile' e `a` quella già esistente. Ci serve solo la nuova variante

```
\cs_generate_variant:Nn
  \example_setvar:Nn
  { cv }
```

e potremo definire

```
\NewDocumentCommand{\copyvar}{mm}
 {
  \example_setvar:cv
    { l_example_var_#1_tl }
    { l_example_var_#2_tl }
 }
```

Ora abbiamo a disposizione un'altra funzione, cioè `\example_setvar:cv` che prende due argomenti tra graffe. Il secondo è elaborato come `c` e produce un token simbolico che dev'essere una variabile della quale si userà *il contenuto* come se fosse un argomento tra graffe dato alla funzione principale.

Lascio come facile esercizio su `\expandafter` scrivere il codice corrispondente in LaTeX tradizionale (suggerimento: si usino `\let` e due `\expandafter` in posizioni opportune).

Si può anche riassumere la definizione di entrambe le varianti:

```
\cs_generate_variant:Nn
  \example_setvar:Nn
  { cn, cv }
```

La variante `v` è un caso speciale della `V` che è molto simile e la maiuscola ci ricorda che l'argomento dev'essere un singolo token (il nome di una variabile, per la precisione) senza graffe. Supponiamo di avere una funzione che faccia qualcosa con il suo argomento:

```
\cs_new:Nn \example_foo:n { -- #1 -- }
```

(solo un esempio stupido per illustrare il concetto). Tuttavia, in alcuni casi ci serve passare alla funzione qualcosa che è stato immagazzinato in una variabile `tl`. Niente di più semplice perché possiamo dire

```
\cs_generate_variant:Nn
  \example_foo:n
  { V }
```

e questo ci permette di chiamare

```
\example_foo:V \l_tmpa_tl
```

Se, per esempio, `\l_tmpa_tl` contiene `abc`, la chiamata `\example_foo:V \l_tmpa_tl` darà lo stesso risultato di `\example_foo:n {abc}`.

Occorre ricordare che le varianti non nascono da sole senza che prima le generiamo. Le funzioni del nucleo sono già fornite con molte varianti che si sono dimostrate utili e che sono elencate insieme alla funzione principale in `interface3.pdf`. Che succede se non sappiamo se una variante è già stata definita? Nessun problema! La nuova generazione verrebbe silenziosamente ignorata e, anche se non lo fosse, il problema non esisterebbe comunque, perché le varianti sono generate in modo uniforme.

Potremmo anche evitare di generare varianti. Per esempio, il mestiere di `\example_setvar:cv` potrebbe essere assunto da

```
\exp_args:Ncv \example_setvar:Nn
```

(che, di fatto, è come la variante è definita), ma non c'è alcuna ragione di complicarci la vita così. Le implementazioni moderne di TeX e soci sono dotate di abbondante memoria e i tempi in cui la memoria era *molto* scarsa e trucchi che risparmiavano anche pochi token erano importanti se li ricordano solo i dinosauri come Frank, David, Chris e io. Ricordo bene quella volta in cui vidi il tremendo messaggio di errore "This can't happen" perché stavo adoperando PiCTeX.

Torniamo alla teoria. Ci sono varianti delle funzioni che definiscono funzioni? Certo che sì! Ci sono occasioni in cui desideriamo definire funzioni il cui nome è deciso durante la compilazione. L'esempio seguente è un po' scemo, ma dovrebbe dare l'idea:

```
\cs_new:cn { example_foo:n } { -- #1 -- }
```

è lo stesso della definizione precedente di `\example_foo:n`, perché il nome, compresa la segnatura, sarà formato prima che la funzione principale `\cs_new:Nn` svolga il suo compito. Si può usare qualsiasi cosa nelle graffe corrispondenti a un argomento `c` purché il risultato finale, dopo l'*espansione completa* consista solo di caratteri.

Oh! Espansione! Che cos'è? Siate pazienti. C'è ancora qualcosa da discutere prima.

3.2 Locale, globale e ⟨*extra*⟩

Tutti dovrebbero sapere qualcosa su *locale* e *globale*. In LaTeX, se definiamo un comando all'interno di un ambiente, la definizione è locale all'ambiente stesso e sparirà quando finisce. Altre assegnazioni di significati sono invece globali: le operazioni sui contatori, per esempio. Non ci interessa discutere in dettaglio la differenza tra locale e globale, ma ci sono aspetti della questione che sono rilevanti per le funzioni.

Tutte le dichiarazioni `\cs_new`⟨*extra*⟩`:Nn` sono sempre *globali*. Anche se eseguite all'interno di un gruppo, il loro effetto sarà presente ai livelli superiori. Inoltre quelle funzioni controllano se la funzione sia già definita e, nel caso, emetteranno un messaggio di errore. La parte ⟨*extra*⟩ sarà descritta fra poco. Va anche notato che anche l'allocazione di variabili (ma non l'impostazione) è sempre globale: l'istruzione

```
\tl_new:Nn \l_example_foo_tl
```

definisce la variabile a tutti i livelli e protesterà se la variabile esiste già.

Però qualche volta ci serve una funzione ausiliaria che sia *definita localmente*, che non abbia un compito fisso e vada ridefinita in base al contesto. Per queste funzioni c'è la famiglia

`\cs_set`⟨*extra*⟩`:Nn`

La sintassi è esattamente la stessa e anche le varianti.

Quale preferire? La famiglia `new` or la `set`? Facile: la prima, a meno che la funzione non sia una che *deve* cambiare definizione secondo il contesto. Raramente, direi anche mai, si definisce una funzione di alto livello con `set`.

La natura stessa del linguaggio invita i programmatori a scrivere codice "a strati". Per esempio, avremmo anche potuto definire

```
\NewDocumentCommand{\setvar}{mm}
 {
  \tl_clear_new:c
    { l_example_var_#1_tl }
  \tl_set:cn
    { l_example_var_#1_tl }
    { #2 }
 }
```

senza introdurre `\example_setvar:Nn`. Scoraggio questo stile di programmazione: il nostro code dovrebbe *appoggiarsi* a expl3 e fornire API che il programmatore impiega. Come detto prima, non c'è ragione di risparmiare una funzione, tanto più se consideriamo che quando la nostra API è disponibile, possiamo facilmente generarne varianti per compiti particolari.

Qual è la lista completa degli 'extra' dopo `\cs_new` o `\cs_set`? Eccola:

```
\cs_new:Nn
\cs_new_protected:Nn
\cs_new_nopar:Nn
\cs_new_protected_nopar:Nn
\cs_set:Nn
\cs_set_protected:Nn
\cs_set_nopar:Nn
\cs_set_protected_nopar:Nn
\cs_gset:Nn
\cs_gset_protected:Nn
\cs_gset_nopar:Nn
\cs_gset_protected_nopar:Nn
```

Dovreste già avere un'idea dello scopo di `protected`: fa sì che la funzione non sia espansa quando la *piena espansione* è forzata. In particolare, una funzione `protected` non può essere adoperata in un argomento di tipo `c`, perché non sarebbe espansa e non è un carattere. La varietà `nopar` proibisce `\par` negli argomenti della funzione (quando è chiamata). A meno di non essere in situazioni speciali in cui `\par` non è sensato negli argomenti, si eviti la varietà *nopar* (ma non ci sono leggi che lo impongano).

La famiglia `gset` è quasi lo stesso di `new`, ma senza il controllo se la funzione da definire esista già.

Per i guru di TeX che leggono queste note„ `new` and `gset` usano `\gdef`, mentre `set` usa `\def`; il prefisso `\long` è aggiunto tranne che nella varietà `nopar`.

Quali sono le varianti disponibili? Questa è la lista completa delle segnature disponibili per tutte le funzioni elencate sopra:

```
Nn  cn  Nx  cx
```

Che fa la misteriosa `x`? Dovrebbe ricordare *fully e**x**panded*. Ci sarà una sezione sull'argomento.

3.3 Parametri

Qualcuno potrebbe protestare dicendo che ha visto modi diversi di definire funzioni e avrebbe ragione. Infatti *c'è* un'intera altra famiglia come quella descritta, ma in cui la segnatura ha una bizzarra `p` tra `N` e `n`:

```
\cs_new:Npn
\cs_new_protected:Npn
\cs_new_nopar:Npn
\cs_new_protected_nopar:Npn
\cs_set:Npn
```

```
\cs_set_protected:Npn
\cs_set_nopar:Npn
\cs_set_protected_nopar:Npn
\cs_gset:Npn
\cs_gset_protected:Npn
\cs_gset_nopar:Npn
\cs_gset_protected_nopar:Npn
```

Il tipo `p` è riservato a queste funzioni e alle loro varianti

```
Npn  cpn  Npx  cpx
```

e sta per ***p**arameter text*. Le righe di codice seguenti sono del tutto equivalenti:

```
\cs_new:Nn \example_usevar:n {...}
\cs_new:Npn \example_usevar:n #1 {...}
```

Lo stesso per le altre funzioni. Nel secondo caso, il *parameter text* è scritto esplicitamente. Ricordiamo che quando sono in vigore le convenzioni di expl3 gli spazi sono ignorati, quindi per due parametri possiamo avere

```
\cs_new:Nn \example_foo:nn {...}
\cs_new:Npn \example_foo:nn #1#2{...}
\cs_new:Npn \example_foo:nn #1#2 {...}
\cs_new:Npn \example_foo:nn #1 #2{...}
\cs_new:Npn \example_foo:nn #1 #2 {...}
```

e le ultime quattro righe sono del tutto equivalenti. Personalmente preferisco la prima che mi appare 'più logica', altri preferiscono il secondo metodo. Attenzione! Il secondo metodo non controlla la coerenza della segnatura con il *parameter text* e permette anche segnature 'sbagliate', ma questo fatto non deve essere sfruttato: LaTeX non protesterà se scrivete

```
\cs_new_protected:Npn
  \example_setvar:cn #1 #2
  {...}
```

ma questo non significa che possiate evitare la procedura in due passi di prima definire `\example_setvar:Nn` e di generare poi la variante. Scrivere la definizione in quel modo sarebbe *sbagliato*. La seconda famiglia di funzioni (quelle con la `p`) ammette perfino che non ci sia la segnatura e quindi può essere adoperata per definire comandi a livello utente, sebbene il metodo con `\NewDocumentCommand` e simili sia raccomandato.[3]

Il metodo con `p` è necessario quando il *parameter text* è 'non standard', nel senso che stiamo definendo una funzione con *argomenti delimitati* e in questo caso la segnatura dovrebbe essere `:w`. Se vogliamo una funzione che imposti tre variabili all'anno, mese e giorno partendo da una data nel formato ISO come 2020-10-15, potremmo fare così:

3. expl3 può anche essere usato con plain TeX e, in tal caso, questo è l'unico modo per definire comandi a livello utente.

```
\int_new:N \l_example_year_int
\int_new:N \l_example_month_int
\int_new:N \l_example_day_int
\cs_new_protected:Nn \example_setdate:n
 {
  \__example_setdate:w #1 \q_stop
 }
\cs_new_protected:Npn
  \__example_setdate:w #1-#2-#3 \q_stop
 {
  \int_set:Nn \l_example_year_int { #1 }
  \int_set:Nn \l_example_month_int { #2 }
  \int_set:Nn \l_example_day_int { #3 }
 }
```

Qui introduco un'altra utile convenzione. Se il ⟨*prefisso*⟩ è preceduto da un doppio underscore, la funzione va considerata di livello più basso delle altre e non dovrebbe mai essere chiamata al di fuori degli usi specifici da parte delle funzioni 'normali' (senza il doppio underscore). L'idea è che le funzioni normali sono 'l'interfaccia del programmatore' mentre le altre sono ausiliarie e la loro implementazione non dovrebbe importare. La distinzione, quando si scrive codice per uso personale non è così importante: lo è per chi scrive pacchetti, però. Le funzioni normali (senza il doppio underscore) *possono* essere adoperate da altri pacchetti; al contrario, nessuno dovrebbe far conto che le funzioni di livello inferiore (con il doppio underscore) siano addirittura definite in versioni successive del pacchetto.

Questo dovrebbe chiarire perché il codice precedente divide il lavoro a due livelli; abbiamo la funzione di alto livello `\example_setdate:n` che si appoggia a una di livello inferiore che faccia il lavoro sporco. L'autore del pacchetto potrebbe scoprire in seguito metodi migliori per quello scopo, ma questo avrebbe influenza solo sul livello inferiore e non sulla funzione principale che si potrà adoperare sempre e allo stesso modo. Magari la definizione di `\example_setdate:n` cambierà in futuro, ma questo non avrà effetto sul codice che la adoperi.

4 Espansione

Non si può capire a fondo TeX senza avere un minimo di conoscenza del suo funzionamento. Nei linguaggi di programmazione funzionali, se `g` è una funzione di una variabile che restituisce un array di tre dati, mentre `f` è una funzione di tre variabili, un'istruzione

```
f(g(x))
```

sarebbe permessa; lo è dal punto di vista della matematica, il particolare linguaggio di programmazione potrebbe richiedere qualche aggiustamento. Questo non succede con TeX che procede dall'esterno verso l'interno e non viceversa.

Se abbiamo la funzione con un argomento `\example_a:n` che restituisce tre liste di token tra graffe e la funzione `\example_b:nnn` che prende tre argomenti, una chiamata come

```
\example_b:nnn { \example_a:n {x} }
```

fallirebbe in modo miserabile. TEX funziona così e nessun codice, per quanto perfezionato e furbo può cambiare la situazione. La funzione esterna cercherà tre argomenti: si trova una sola lista di token tra graffe e si arrangerà in altro modo, con risultati disastrosi. Per fare un esempio, supponiamo che a `\example_a:n` possa essere dato come argomento una data in formato ISO format e che da `2020-10-15` restituisca `{2020}{10}{15}`, mentre `\example_b:nnn` prende tre argomenti e produce la data in un formato diverso, diciamo "giorno 'nome del mese' anno"; in questo caso dovrebbe stampare "15 ottobre 2020".

Ci serve un approccio indiretto se vogliamo permettere di fornire una data ISO alla funzione che stampa la data nel formato tradizionale. Vediamo come si può definire la funzione generale:

```
\cs_new:Nn \example_date:nnn
 { % #1 = year, #2 = month, #3 = day
  #3~
  \int_case:nn { #2 }
   {
    {1}{gennaio}
    {2}{febbraio}
    ...
    {12}{dicembre}
   }
  ,~#1
 }
```

Il codice non è completo, dovrebbe essere chiaro come riempire il buco. La funzione `\int_case:nn` compara il suo primo argomento, un intero, con la lista data come secondo argomento che consiste di coppie di oggetti tra graffe; il primo oggetto è un intero, il secondo il testo da restituire se gli interi coincidono. Notiamo che `~`, nell'ambiente di programmazione `expl3` è un semplice spazio.

Abbiamo anche la funzione che prende una data nel formato ISO come `2020-10-15` e restituisce `{2020}{10}{15}`; potrebbe essere

```
\cs_new:Nn \__example_isodate:n
 {
  \__example_isodate:w #1 \q_stop
 }
\cs_new:Npn
  \__example_isodate:w #1-#2-#3 \q_stop
 {
  {#1}{#2}{#3}
 }
```

L'input della seconda funzione è diviso ai trattini e al terminatore, cioè stiamo adoperando *argomenti delimitati*, dettaglio su cui non insisto: ci basta sapere che la chiamata `\__example_isodate:n {2020-10-15}` in qualche modo restituirà `{2020}{10}{15}` nel flusso di input di TEX.

Come le combiniamo? Ci sono parecchi modi, ma tutti richiedono di capire il concetto di *piena espansione*. TEX conosce solo le macro; quando ne trova una, sa quanti argomenti vuole e li cerca nel flusso di input; quando li ha trovati, sostituisce l'intero insieme di token con il testo di sostituzione della macro.

Il problama primario è che il più delle volte non abbiamo la minima idea di quanti passi di questo processo di espansione ci sia bisogno per passare da `\__example_isodate:n {2020-10-15}` a `{2020}{10}{15}`. In questo caso non sarebbe difficile contarli, ma è solo un esempio semplice. Se lo sapessimo, un'opportuna catena di `\expandafter` basterebbe, ma sarebbe facilissimo commettere errori e non garantirebbe nulla se per caso l'implementazione della funzione di livello inferiore cambiasse.

Vorremmo dunque che `\__example_isodate:n` arrivasse al risultato finale in una sola mossa. Ecco un modo:

```
\exp_last_unbraced:Ne
  \example_date:nnn
  { \__example_isodate:n {2020-10-15} }
```

La prima funzione ha lo scopo di produrre la *piena espansione* del suo secondo argomento e inserisce il risultato nel flusso di input senza graffe attorno. Ci sono altri modi: uno è di definire una funzione ausiliaria

```
\cs_new:Nn \example_date:n
 {
  \__example_date:Ne
    \example_date:nnn
    { \__example_isodate:n { #1 } }
 }
\cs_new:Nn \__example_date:Nn
 {
  #1 #2
 }
\cs_generate_variant:Nn
  \__example_date:Nn { Ne }
```

e così `\example_date:n {2020-10-15}` produrrebbe il risultato voluto.

Ci sono certamente modi migliori per gestire le date, ma l'idea era di presentare come sfruttare le varianti che producono piena espansione.

Ce ne sono di tre tipi: `e`, `x` e `f`. L'ultima è la più 'restrittiva' perché esegue un'espansione ricorsiva dei token non appena appaiono nel flusso di input e termina al primo token non espandibile che trova. Nonostante questa limitazione, ha parecchi usi.

Il tipo `x` è al giorno d'oggi meno importante perché tutti i motori TEX possiedono la primitiva

`\expanded` che è essa stessa espandibile. expandable. Solo TEX 'originale Knuthiano' (quello che si lancia con `tex` dalla linea di comando) non ce l'ha, perché è mantenuto senza estensioni, secondo i desideri di Knuth.

Che fa `\expanded`? Fa essenzialmente lo stesso di `\edef`, con la differenza che non viene definita una macro. L'argomento subisce la piena espansione ricorsiva che *non* termina quando si trova un token espandibile: questo viene superato e lasciato com'è e si procede sul successivo, fino al termine della lista di token passata inizialmente. Il risultato va nel flusso di input (tutto insieme e senza graffe).

4.1 Piena espansione con e

Il tipo di argomento `e` dice a LATEX che prima deve eseguire la piena espansione e poi passare il risultato alla funzione principale. Questo è molto importante se per caso l'argomento contiene una variabile di cui vogliamo adoperare il valore al momento in cui la funzione è chiamata.

Ecco un esempio tratto da una domanda posta su TEX.StackExchange.[4] La domanda riguarda aggiungere a una 'nota finale' (*endnote*, pacchetto `endnote`) il numero di pagina dove la nota finale è richiamata. Chiaramente ci serve `\pageref` attraverso una `\label` generata automaticamente:

```
\NewDocumentCommand{\MyEndNote}{m}
 {
  \polyv_myendnote:ne
    { #1 }
    { \int_eval:n { \arabic{endnote}+1 } }
 }

\cs_new_protected:Nn \polyv_myendnote:nn
 {
  \endnote
   {
    #1~(page\nobreakspace
    \pageref{#2:endnote})
   }
  \label{\arabic{endnote}:endnote}
 }
\cs_generate_variant:Nn
  \polyv_myendnote:nn
  { ne }
```

Qual è il problema da risolvere? Il numero della nota finale è incrementato solo dopo che il testo della nota è stato elaborato, in modo che una `\label` successiva si riferisca al numero giusto. Quindi non possiamo usare

```
\pageref{\arabic{endnote}:endnote}
```

perché questo si riferirebbe alla nota *precedente*. Quindi la funzione interna impiega l'espansione `e` in modo da generare il successore del valore attuale del contatore `endnote`. Senza questa piena espansione, tutte le note finali dichiarate con `\MyEndNote` conterrebbero l'equivalente di

```
\pageref{%
  \int_eval:n{\arabic{endnote}+1}:endnote
}
```

e perciò il risultato finale sarebbero riferimenti incrociati indefiniti perché `\arabic{endnote}` fornirebbe l'*ultimo* valore del contatore. Se l'ultima nota finale avesse il numero 10, otterremmo `\pageref{11:endnote}`. Invece, con la piena espansione, viene usato il valore al momento della chiamata. Alla prima nota il contatore ha il valore 0, così il risultato sarà lo stesso di

```
\endnote{The text of the endnote
  (page\nobreakspace\pageref{1:endnote})}
\label{1:endnote}
```

Avremmo anche potuto usare `f` o `x` per questa particolare applicazione, ma `e` è il più efficiente dei tre. La differenza con `x` è che le funzioni che usano questa variante non sono espandibili, quindi devono essere del tipo `protected`. Infatti il processo richiede due passi: prima viene impostata una variabile `tl` con

```
\tl_set:Nx \l__exp_internal_tl {...}
```

che sfrutta internamente il buon vecchio `\edef`, poi la funzione è definita adoperando il valore della variabile.

L'introduzione della piena espansione `e` è stato un grande avanzamento perché permette cose che prima erano quasi impossibili.

Ci sono comunque impieghi utili di `x`: per esempio non ci sono varianti come `\cs_new:Ne` che sarebbe *meno* efficiente di `\cs_new:Nx` (che è esattamente `\edef`).

5 Un'altra utile variante

Nell'elenco dei tipi di argomento ci sono `V` e `v` su cui è stato dato qualche cenno. È ora di discutere il primo tipo in maggiore profondità.

Il tipo `v` è essenzialmente lo stesso e aggiunge solo la possibilità di costruire il nome della variabile durante la compilazione. Il tipo principale è `V`.

Supponiamo di avere la nostra funzione preferita che divide una data ISO nelle sue parti e la restituisce in forma leggibile. Chiamiamola `\example_date:n`, non ci interessa la sua implementazione.

Supponiamo ora che una data sia stata immagazzinata in una variabile `tl`. Siccome si tratta di una macro sotto travestimento, ai primordi di expl3 il modo di agire era

```
\cs_generate_variant:Nn
  \example_date:n { o }
```

4. https://tex.stackexchange.com/a/438715/. Il codice nella risposta adopera `f` perché `e` non era ancora disponibile.

e si doveva chiamare

```
\example_date:o { \l_tmpa_tl }
```

Funziona, ma ha il difetto di dipendere dalla conoscenza dell'implementazione delle variabili `tl` e non è generalizzabile ad altri tipi di variabili. La variante `o` esegue un singolo passo di espansione all'interno dell'argomento tra graffe; fa il suo dovere con una `tl`, ma fallirebbe in modo spettacolare con una variabile `fp` (che contiene la rappresentazione di un numero in virgola mobile).

Il metodo migliore e raccomandato è

```
\cs_generate_variant:Nn
  \example_date:n { V }
```

e la nuova funzione va chiamata come

```
\example_date:V \l_tmpa_tl
```

L'effetto è di passare alla funzione principale il *contenuto* della variabile, con le graffe al loro posto. Dopo

```
\tl_set:Nn \l_tmpa_tl { 2020-10-15 }
```

la chiamata `\example_date:V \l_tmpa_tl` sarà equivalente a `\example_date:n {2020-10-15}`.

Ecco un esempio con un tipo diverso di variabile, qui `int`. Vogliamo passare la rappresentazione decimale del valore e al risultato vanno aggiunti zeri a sinistra in modo da avere quattro cifre. Definiamo la funzione principale che genera tanti zeri quanti ne mancano, contando il numero di cifre del numero passato; poi generiamo la variante `V` e impostiamo una variabile `int` a un valore

```
\cs_new:Nn \example_pad:n
 {
  \prg_replicate:nn
    { 4 - \tl_count:n { #1 } }
    { 0 }
  #1
 }
\cs_generate_variant:Nn
  \example_pad:n { V }

\int_set:Nn \l_tmpa_int { 43 }
\example_pad:V \l_tmpa_int
```

Otterremo 0043. Forse non sembra così interessante, ma se usiamo il cugino `v` possiamo trasformare in

```
\cs_generate_variant:Nn
  \example_pad:n { v }

\example_pad:v { c@page }
```

sapendo, ovviamente che `\c@page` è il nome in LaTeX del registro che contiene il numero di pagina: si può immediatamente pensare a un'applicazione. Qui sfruttiamo il fatto che i contatori TeX sono proprio come le variabili `int` di expl3. Tramite le varianti `V` o `v` stiamo passando alla funzione principale il valore corrente come lista di cifre e possiamo contare quante ce ne sono, il che sarebbe impossibile con il valore 'astratto'.

Quali tipi di variabili possono essere adoperate in questo modo? Parecchie: `tl`, `int`, `fp`, ma anche `clist` e altre più esoteriche. In sostanza, tutte le variabili che possono restituire un output sensato. Non ce lo possiamo aspettare da variabili `seq` o `prop` e con queste otterremmo errori.

Ho trovato talvolta utile definire la variante `\cs_set:NV` a `\cs_set:Nn` per poter adoperare una variabile `tl` che è stata impostata con il testo di sostituzione desiderato, ma modificato con l'aiuto delle espressioni regolari.[5] Si noti che non è possibile definire la variante `\cs_set:NpV`, perché `\cs_generate_variant:Nn` accetta funzioni la cui segnatura contiene solo i caratteri `N` o `n`.

6 Vero o falso?

Ci sono altri due tipi di argomento interessanti: `T` e `F`. Il titolo della sezione dovrebbe suggerire che sono correlati a verità e falsità.

Giusto! Sono specificatori di argomento per le segnature delle funzioni *condizionali*. Esempio:

```
\int_compare:nTF
```

è una funzione che prende tre normali argomenti tra graffe; il primo è una relazione numerica tra interi che viene esaminata, il secondo il codice da eseguire se la relazione è vera, il terzo il codice da eseguire se la relazione è falsa. Quindi

```
\int_compare:nTF { 0<1 } { A } { B }
\int_compare:nTF { 0>1 } { A } { B }
```

stamperà, rispettivamente, A e B. Perché non è, più semplicemente, `\int_compare:nnn`? In realtà era proprio così nelle prime versioni di expl3, ma si comprese che gli specificatori `T` e `F` sono migliori: possiamo anche avere

```
\int_compare:nT
\int_compare:nF
```

per i casi in cui non ci sia da eseguire nulla nei casi falso e vero rispettivamente. Certo,

```
\int_compare:nF { ⟨relation⟩ } { B }
\int_compare:nTF { ⟨relation⟩ } { } { B }
```

sono del tutto equivalenti, ma il primo mostra più chiaramente che non vogliamo fare nulla se la relazione è vera. Con la segnatura `:nnn` sarebbe comunque necessario avere argomenti vuoti. Inoltre, la presenza di `T` or `F` (o entrambi) ci fa vedere bene che la funzione è condizionale.

5. https://tex.stackexchange.com/a/355576/

Tutte le funzioni condizionali del nucleo sono disponibili con `TF`, `T` or `F`; alcune funzioni pseudo-condizionali hanno perfino la versione senza entrambi. Ne troviamo un esempio in `interface3.pdf` dove ci sono `\str_case:nn` e anche `\str_case:nn`*`TF`*. La strana notazione *`TF`* significa che sono disponibili le tre combinazioni `TF`, `T` e `F`.

Perché pseudo-condizionale? È chiaro che `\str_case:nn` non è un condizionale, ma la versione estesa permette di eseguire qualcosa in più se c'è o non c'è un *match*. Probabilmente si impiega di più la versione `\str_case:nnF` per eseguire qualcosa in caso di *no match* che può essere un messaggio di errore o qualche *default*.

Va notato che per i condizionali (anche pseudo) le varianti vanno generate con

```
\prg_generate_conditional_variant:Nnn
```

invece che con `\cs_generate_variant:Nn`. Per esempio se pensiamo di immagazzinare relazioni numeriche per `\int_compare:n`*`TF`* in una variabile `tl`, il modo corretto di generare la variante è

```
\prg_generate_conditional_variant:Nnn
  \int_compare:n { V } { p, TF, T, F }
```

Questo genera tutte le varianti

```
\int_compare:VTF
\int_compare:VT
\int_compare:VF
```

e anche la *forma predicativa*

```
\int_compare_p:V
```

che si può usare nelle espressioni booleane. Ma questo va oltre gli scopi di questo articolo.

▷ Enrico Gregorio
Dipartimento di Informatica
Università di Verona
enrico dot gregorio at univr dot it

Questa rivista è stata prodotta
dal Gruppo Utilizzatori Italiani di TEX
usando esclusivamente software libero.

Versione elettronica per la diffusione via web.

ArsTEXnica – Call for Paper

La rivista è aperta al contributo di tutti coloro che vogliano partecipare con un proprio articolo. Questo dovrà essere inviato alla redazione di ArsTEXnica, per essere sottoposto alla valutazione di recensori entro e non oltre il 14 Febbraio 2021. È necessario che gli autori utilizzino la classe di documento ufficiale della rivista; l'autore troverà raccomandazioni e istruzioni più dettagliate all'interno del file d'esempio (.tex).

Gli articoli potranno trattare di qualsiasi argomento inerente al mondo di LATEX e non dovranno necessariamente essere indirizzati ad un pubblico esperto. In particolare tutorial, rassegne e analisi comparate di pacchetti di uso comune, studi di applicazioni reali, saranno bene accetti, così come articoli riguardanti l'interazione con altre tecnologie correlate.

Di volta in volta verrà fissato, e reso pubblico sulla pagina web `http://www.guitex.org/arstexnica/`, un termine di scadenza per la presentazione degli articoli da pubblicare nel numero in preparazione della rivista. Tuttavia gli articoli potranno essere inviati in qualsiasi momento e troveranno collocazione, eventualmente, nei numeri seguenti.

Chiunque, poi, volesse collaborare con la rivista a qualsiasi titolo (recensore, revisore di bozze, grafico, etc.) può contattare la redazione all'indirizzo arstexnica@guitex.org.

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 30, Ottobre 2020

00030
9 771828 236001