GUIT

TEXnica ArS

Rivista italiana
di TEX, LATEX e
tipografia digitale

32 • 2021

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 e 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 da Donald Knuth.

Le uscite hanno cadenza semestrale e sono pubblicate nei mesi di Aprile e Ottobre. In particolare, la seconda uscita dell'anno contiene gli Atti del Convegno Annuale (GUIT*meeting*).

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

arstexnica@guitex.org.

Pubblicare su ArsTEXnica

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.

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 2021

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

GUIT *meeting* 2021

DICIOTTESIMO CONVEGNO NAZIONALE SU TEX, LATEX E TIPOGRAFIA DIGITALE

16 ottobre 2021

Firenze

online

Programma del convegno

10:00 Messaggio di benvenuto. Inizio dei lavori.

10:10 *Functions in TEX and Elsewhere*. Jean Michel Hufflen.
10:40 *UniFiTh: una classe per le tesi dell'Università di Firenze*. Francesco Biccari.
11:10 *Personalizzare TEXworks*. Filippo Vomiero.

11:40 Discussione sul futuro del Gruppo Utilizzatori Italiani di TEX.
12:00 Chiusura dei lavori.

La partecipazione è libera e gratuita, si consiglia di registrarsi entro il 15 ottobre.
Registrazione online: www.guitex.org/home/meeting

Gruppo Utilizzatori Italiani di TEX ⋆ GUIT

www.guitex.org

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 32, Ottobre 2021

Gruppo Utilizzatori Italiani di TEX

Editoriale

Francesco Biccari

Cari lettori, 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.

Il GUIT meeting di quest'anno avrebbe dovuto tenersi il 16 ottobre 2021 presso il Dipartimento di Fisica e Astronomia dell'Università degli Studi di Firenze. Purtroppo, a causa del perdurare della pandemia di COVID-19, la prenotazione degli spazi universitari per eventi non strettamente legati alla didattica è ancora soggetta a regole molto stringenti e quindi siamo stati obbligati a svolgere il GUIT meeting in modalità telematica con la diretta in *streaming* su YouTube. Nonostante tutto, la conferenza si è svolta senza intoppi. Per questo devo ringraziare Giovanni Marcotullio, membro del Comitato Direttivo del GUIT, e Giovanna Pacini, vicepresidente dell'associazione Caffè Scienza di Firenze. L'e aiuto che mi hanno dato nell'organizzazione e gestione del GUIT meeting è stato fondamentale.

L'incontro prevedeva tre interventi: potete trovare le presentazioni in formato pdf e i video della conferenza sulla pagina del GUIT meeting[1].

Prima di darvi un'anteprima di questi tre contributi, vorrei ricordare la GUIT *challenge* che era stata pubblicata nel numero precedente. La sfida prevedeva che i partecipanti si cimentassero nel preparare un pacchetto per elaborare e mettere in grafico dei dati sperimentali forniti in un file esterno. Non essendo arrivata alcuna soluzione, abbiamo deciso di posticipare la scadenza al giorno 31 marzo 2022. Il vincitore verrà annunciato nel prossimo numero di ArsTEXnica: vincerà il volume *TEX by topic* avvolto nella famosa carta da regalo del GUIT.

Ma vediamo quali sono i tre contributi presentati al GUIT meeting che troverete in questo numero.

Apriamo con la presentazione diUniFiTh una nuova classe LATEX preparata dal sottoscritto, UniFiTh, che permette di comporre una tesi secondo lo stile previsto dall'Università degli Studi di Firenze. Tutti ricordiamo quanto tempo abbiamo perso per rifinire i dettagli estetici delle nostre tesi in LATEX invece di focalizzarci sui contenuti. Nonostante le innumerevoli guide in circolazione oggigiorno, tutto questo non è cambiato. Nel 2019 ho quindi realizzato UniFiTh, una classe per la composizione delle tesi dell'Università degli Studi di Firenze, università nella quale lavoro. In concomitanza con il rilascio dell'ultima versione del 1 ottobre 2021, ho deciso di presentare UniFiTh al GUIT meeting di quest'anno. Nel primo articolo di questo numero troverete quindi una guida all'uso di questa classe, corredata di molti esempi. Ci saranno anche alcuni estratti commentati del codice della classe che potrebbero essere di interesse generale. Infine, verranno affrontati due problemi, non direttamente legati alla classe, ma che si presentano frequentemente durante la realizzazione di una tesi: la dimensione troppo grande del file pdf della tesi e la generazione di un pdf archiviabile.

Filippo Vomiero è l'autore del secondo articolo: affronta il problema di come personalizzare TEXworks, l'editor incluso di default in tutte le distribuzioni TEX più famose. TEXworks

1. https://www.guitex.org/home/it/guit-meeting-2021

è un editor con pochi fronzoli, tuttavia, nonostante a prima vista non lo sembri, è anche estremamente personalizzabile. Il contributo illustra nello specifico come personalizzare alcuni aspetti di TEXworks: il tema dell'interfaccia, la colorazione della sintassi, i dizionari e gli strumenti di composizione. L'idea non è però solo quella di modificare TeXworks in base ai propri gusti estetici ma anche permettere di intervenire su aspetti più importanti come il miglioramento della leggibilità per gli ipovedenti.

Il terzo e ultimo articolo è di carattere più generale. L'autore Jean Michel Hufflen presenta in maniera critica alcuni aspetti della sintassi e del comportamento del linguaggio TEX e dei suoi derivati per quanto riguarda le funzioni. Confronta le definizioni di funzione in matematica e in alcuni linguaggi di programmazione e mostra, in particolare, alcune peculiarità di TEX e LATEX rispetto agli altri linguaggi. Un'analoga analisi viene fatta sul concetto di *tipo* di dato. Infine, viene accennato a come alcune di queste peculiarità vengono mitigate in LATEX3.

Rinnovando l'invito a inviare possibili soluzioni per la GUIT challenge pubblicata nello scorso numero, vi saluto e vi auguro una buona lettura.

Francesco Biccari
Dipartimento di Fisica e Astronomia
Università degli Studi di Firenze
Firenze, Italia
biccari@gmail.com

UniFiTh: una classe per le tesi dell'Università degli Studi di Firenze

Francesco Biccari

Sommario UniFiTh è una classe LaTeX utile alla composizione delle tesi dell'Università degli Studi di Firenze. È basata sulla classe standard book. Oltre allo stile della pagina, UniFiTh fornisce una serie di comandi per la composizione del frontespizio e del retro del frontespizio per dare un aspetto omogeneo e professionale alle tesi. Fornisce infine alcuni comandi utili in campo scientifico. In questo articolo verranno mostrate tutte le funzionalità di UniFiTh e diversi esempi d'uso.

Abstract UniFiTh is a LaTeX class to typeset the theses of the University of Florence. It is based on the standard class book. In addition to pagestyle, UniFiTh provides a series of commands for the typesetting of the titlepage and the back of the titlepage, in order to give a consistent and professional look to the theses. Finally, it provides some useful commands for scientific fields. In this article we show all the features of the UniFiTh class and several usage examples.

1. Introduzione

Dopo molti anni nei dipartimenti universitari dove ho studiato e poi lavorato mi sono accorto che gli studenti spendono molto tempo per rifinire i dettagli estetici dei loro documenti LaTeX invece di focalizzarsi sul contenuto. Questo è contro la filosofia di LaTeX, che è stato creato proprio per sollevare lo scrittore dal lavoro del tipografo. Oltre al tempo speso, i documenti risultanti sono differenti gli uni dagli altri e non sempre si ottengono buoni risultati estetici. Questo ha un impatto negativo sia sul lavoro dello studente sia sull'immagine dell'Università. Per queste ragioni ho pensato che una classe LaTeX per le tesi della mia università fosse una buona idea.

Nel lontano 2010, poco dopo la conclusione del mio percorso di dottorato alla Sapienza Università di Roma, pubblicai la classe Sapthesis, una classe LaTeX che cercava di risolvere i problemi discussi finora.

A fine 2013 cominciai a lavorare presso l'Università degli Studi di Firenze e, nonostante non fossi più uno studente, dopo qualche anno decisi di realizzare una classe analoga per la mia nuova università e così nacque UniFiTh che, almeno inizialmente, ricalcava sia nell'estetica sia nel codice, la classe Sapthesis.

UniFiTh segue le regole del Piano di Comunicazione dell'Università degli Studi di Firenze (AA.VV., c). Per la sua realizzazione devo ringraziare tutti gli studenti dell'Università degli Studi di Firenze che mi hanno mandato dei commenti e in particolare i ricercatori Giacomo Mazzamuto e Lorenzo Pattelli. La prima versione è stata pubblicata il 7 aprile 2019 e fu annunciata anche con un post sul forum del GuIT (Biccari, 2019). Attualmente l'ultima versione è la 1.6, pubblicata sul CTAN il 1 ottobre 2021 (Biccari, 2021) in occasione del GuIT meeting 2021. UniFiTh può essere installata dal gestore dei pacchetti di ogni distribuzione TeX.

In questo articolo verrà presentata la classe UniFiTh e il suo uso. Verranno analizzate brevemente alcune parti del suo codice e infine verranno affrontati due problemi, non direttamente legati alla classe, ma che si presentano frequentemente durante la realizzazione delle tesi: la dimensione troppo grande del file pdf della tesi e la generazione di un pdf archiviabile.

2. Guida all'uso

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

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

Ha diverse opzioni di classe e fornisce svariati comandi, principalmente per la composizione del frontespizio e per il materiale iniziale della tesi (abstract, ringraziamenti, ecc…), ma mette a disposizione anche dei semplici comandi utili a chi scrive tesi in ambito scientifico.

Inizieremo con un esempio minimale dell'uso di UniFiTh. Poi mostreremo le opzioni della classe, i comandi per la composizione del frontespizio e del retro del frontespizio, i comandi appositi per comporre il materiale iniziale e infine vedremo un esempio completo.

2.1. Un esempio minimale

Con il seguente codice minimale si ottiene il risultato riportato nella figura 1, dove sono mostrate le prime due pagine del pdf prodotto dalla compilazione. L'esempio si riferisce a una tesi di dottorato scritta in inglese, ma con piccole variazioni il codice può essere adattato per altri tipi di tesi.

```
% !TeX encoding = UTF-8
% !TeX program = pdflatex
% !TeX spellcheck = en_US

\documentclass[binding=0.6cm]{unifith}

\usepackage{microtype}
\usepackage[english]{babel}
\usepackage[utf8]{inputenc}
\usepackage{hyperref}
\hypersetup{pdftitle={My thesis},
  pdfauthor={Francesco Biccari}}

\title{My thesis}
\alttitle{Titolo seconda lingua}
\author{Francesco Biccari}
\IDnumber{166350}
\course{Corso di Dottorato in\\
  Fisica e Astronomia}
\courseorganizer{Dipartimento di Fisica
  e Astronomia}
\cycle{Ciclo XXXII}
\advisor{Prof. Caio}
```

```
\advisor{Dr. Sempronio}
\AcademicYear{2018/2019}
\thesistype{PhD thesis}
\copyyear{2020}
\authoremail{pippo@pippo.com}

\begin{document}
\frontmatter
\maketitle
...
\end{document}
```

Nella parte iniziale troviamo le righe magiche per il proprio editor, non necessarie ma molto utili. Si possono ovviamente ignorare o modificare opportunamente in base alle caratteristiche del vostro editor.

Nel caricamento di UniFiTh, la prima cosa da notare è l'opzione di classe binding in cui viene specificato il bordo per la rilegatura. Dopo il caricamento di alcuni pacchetti (spesso caricati quando si usa pdfLaTeX), troviamo i comandi per la composizione del frontespizio. Notare il comando `\alttitle` che serve per specificare un titolo alternativo, tipicamente in un'altra lingua rispetto a quella principale.

Gli ultimi tre comandi del preambolo vengono usati per la composizione del retro del frontespizio. Infine il corpo del documento è sezionato con i comandi `\frontmatter`, `\mainmatter`, `\backmatter` come è tipico della classe book su cui UniFiTh si basa. Dopo il comando `\frontmatter` viene composto il frontespizio e il retro del frontespizio in automatico con il classico comando `\maketitle`.

Si faccia attenzione al fatto che quando UniFiTh è installata tramite la vostra distribuzione TeX o scaricata dal CTAN, il logo dell'Università degli Studi di Firenze non è presente. La ragione risiede nell'impossibilità di caricare sul CTAN materiale non liberamente distribuibile. Pertanto sul frontespizio, al posto del logo, ci sarà un rettangolo con un avvertimento e un link. Cliccando sul link potrete scaricare il logo che dovrete poi copiare nella vostra cartella di lavoro (o in altre posizioni rintracciabili dal compilatore).

Il comando `\alttitle` visto nell'esempio riportato sopra è ovviamente opzionale. Un altro comando che può essere usato al suo posto è `\subtitle`, per specificare un sottotitolo al titolo principale. Nella figura 2 è riportato il risultato del codice precedente in cui si è usata la seguente riga di codice al posto di quella col comando `\alttitle`.

```
\subtitle{Sottotitolo}
```

Il comando `\subtitle` ha la precedenza su `\alttitle`. Se quindi vengono usati insieme il comando `\alttitle` viene ignorato.

2.2. Opzioni di classe

Qui di seguito riportiamo l'elenco delle opzioni di classe di UniFiTh.

binding=⟨*length*⟩ (default zero). Abbiamo già visto questa opzione nell'esempio precedente. Serve a specificare lo spazio che verrà usato per la rilegatura.

draft Questa opzione non include le immagini e segna con un quadratino nero i warning di overfull. È l'usuale opzione `draft` delle classi standard di LaTeX.

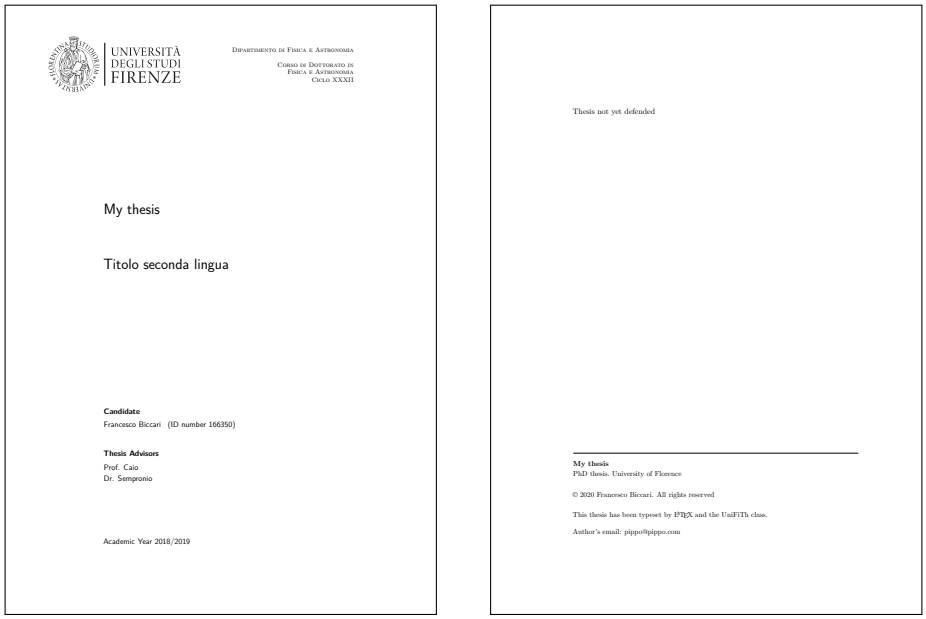

Figura 1. Esempio delle prime due pagine di un documento ottenuto con la classe UniFiTh in cui è stato specificato un titolo alternativo.

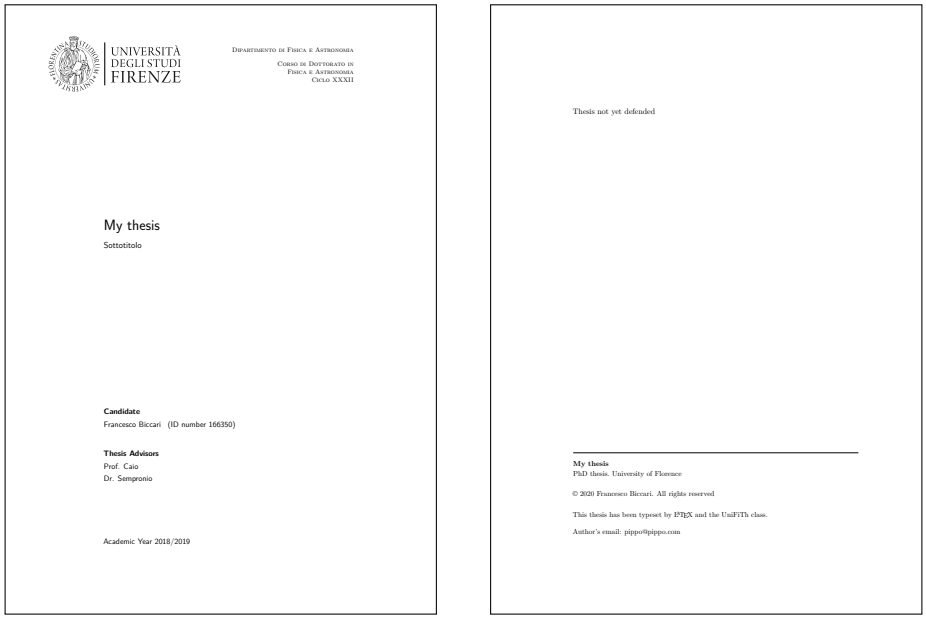

Figura 2. Esempio delle prime due pagine di un documento ottenuto con la classe UniFiTh in cui è stato specificato un sottotitolo.

oneside Per stampare la tesi solo fronte. È l'usuale opzione `oneside` delle classi standard di LaTeX.

twoside (default) Per stampare la tesi fronte/retro. È l'usuale opzione `twoside` delle classi standard di LaTeX.

a4paper/a5paper/b5paper Scegliere uno di questi tre formati di carta. `a4paper` è il valore di default. Nella figura 3 sono riportati, in scala, tre frontespizi ottenuti con lo stesso codice dell'esempio minimale usato precedentemente ma cambiando l'opzione per il formato del foglio.

layout=⟨*a5paper/b5paper*⟩ Se per qualche ragione non è possibile stampare direttamente in A5 o B5, UniFiTh permette di comporre la tesi comunque su A4, riportando dei crocini di taglio per ritagliare a posteriori le pagine A5 o B5. Per fare questo si usa l'opzione a4paper e poi si specifica l'opzione layout al valore desiderato, A5 o B5. Il risultato è mostrato nella figura 4

fem Questa opzione permette di stampare sul frontespizio la dicitura "candidata" anziché "candidato". Ovviamente questa opzione ha effetto solo se si scrive la tesi in italiano.

noexaminfo Elimina tutte le informazioni sull'esame di laurea o di dottorato che vengono riportate nel retro del frontespizio. Per esempio, di default, UniFiTh mostra la dicitura "Thesis not yet defended"/"Tesi non ancora discussa". Oppure dando i comandi `\examdate{...}` and `\examiner{...}`, come vedremo in seguito, sarà possibile stampare le informazioni sull'esame finale. Dando l'opzione noexaminfo queste informazioni non vengono mostrate.

nodefaultfont Dando questa opzione non vengono caricati i pacchetti relativi ai font. È utile se nella tesi vanno usati dei font specifici e non il Latin Modern.

romandiff Questa opzione permette di cambiare il comportamento del comando `\di`. Vedi il paragrafo 2.7.

2.3. I comandi per il frontespizio

Il frontespizio è generato come nelle classi standard di LaTeX, e come abbiamo visto nell'esempio minimale precedente, con il comando `\maketitle`. La composizione del frontespizio si avvale di una serie di comandi, alcuni dei quali opzionali o utili solo per alcuni tipi di tesi. Vediamo quali sono.

`\title{...}` Titolo della tesi.

`\subtitle{...}` Opzionale. Sottotitolo della tesi. Come visto prima.

`\alttitle{...}` Opzionale. Titolo alternativo della tesi. Come visto prima.

`\author{...}` Nome dell'autore della tesi. Vedi l'opzione di classe fem descritta nella sezione precedente.

`\IDnumber{...}` Numero di matricola.

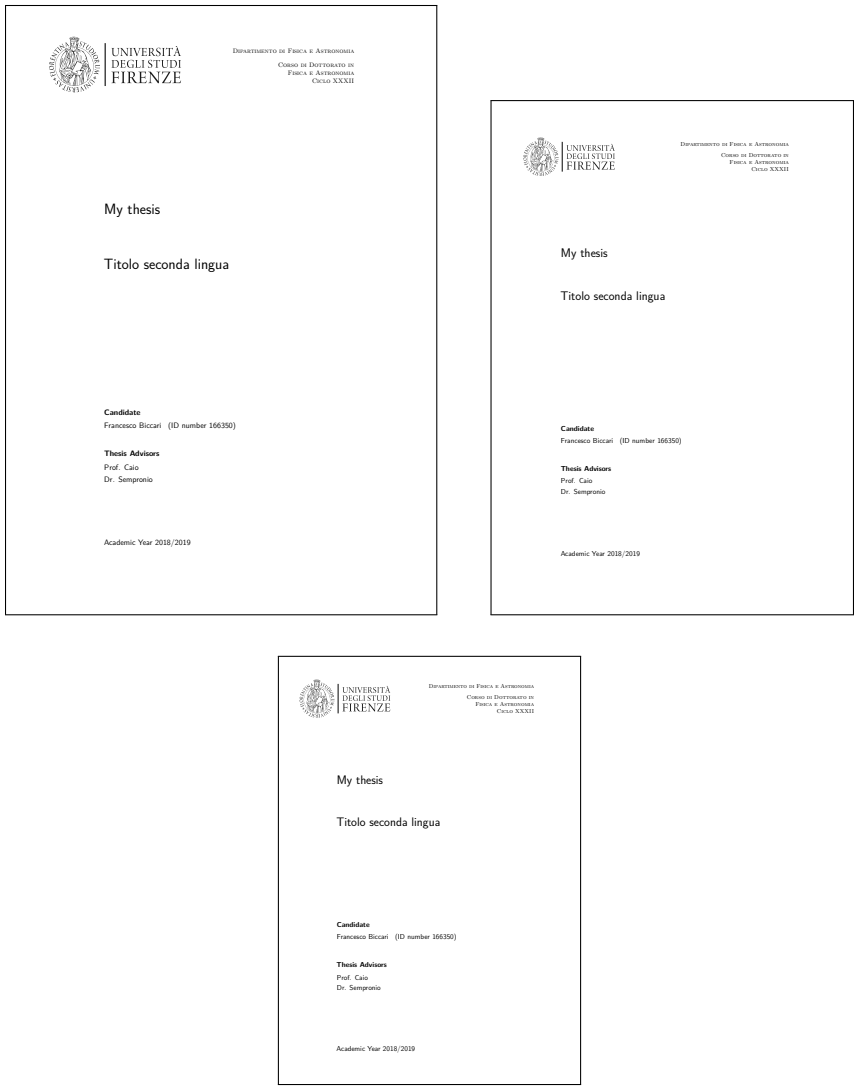

Figura 3. Esempio di frontespizio ottenuto con la classe UniFiTh usando le tre possibili opzioni per il formato del foglio: in alto a destra a4paper, in alto a sinistra b5paper, in basso a5paper. Le dimensione dei fogli sono in scala.

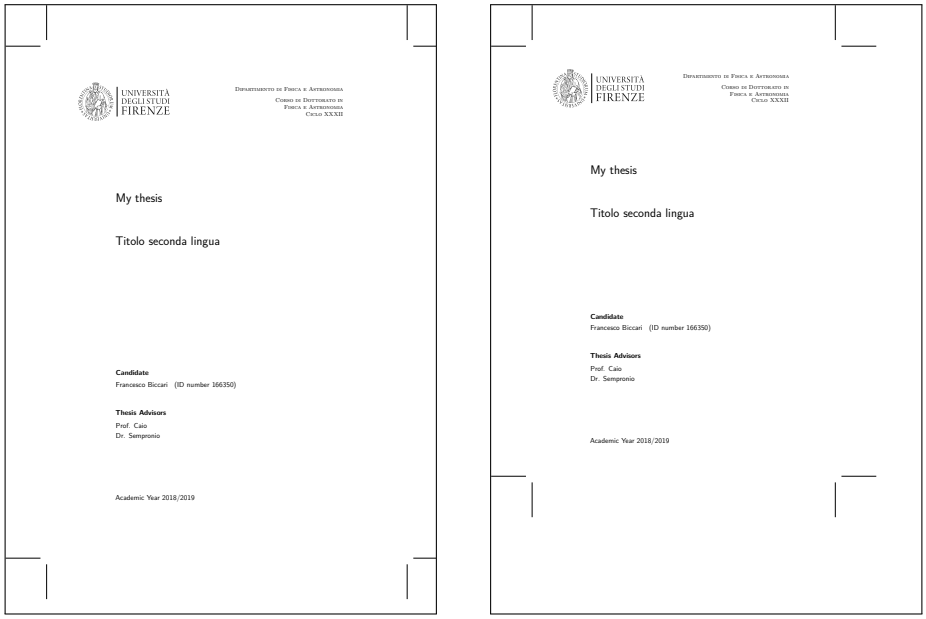

Figura 4. Esempio di frontespizio ottenuto con la classe UniFiTh usando l'opzione di default per il formato del foglio, cioè a4paper, e l'opzione layout. A sinistra layout=b5paper, a destra layout=a5paper. Si notino i crocini di taglio.

`\course{...}` Nome ufficiale del corso di studi (per esempio, Corso di Laurea in Scienze Fisiche e Astrofisiche oppure Corso di Laurea Magistrale in Fisica e Astronomia e così via).

`\cycle{...}` Ciclo di dottorato. Obbligatorio ovviamente solo se si sta scrivendo una tesi di dottorato. Usare i numeri romani: `\cycle{XXII}`

`\courseorganizer{...}` Organizzatore del corso di studi (per esempio Scuola di Scienze Matematiche Fisiche e Naturali, tipico per le lauree, oppure Dipartimento di Fisica e Astronomia, tipico per i dottorati, e così via).

`\AcademicYear{...}` Anno Accademico.

`\advisor{...}` Relatore di tesi. Questo comando può essere ripetuto più volte nel caso in cui si abbiano più relatori.

`\coadvisor{...}` Opzionale. Correlatore della tesi. Può essere ripetuto più volte come il comando `\advisor`.

`\customcoadvisorlabel{...}` Opzionale. Specifica l'etichetta sopra la lista dei correlatori nel caso in cui la scritta di default, "Correlatore", non sia adatta. Per esempio se si vuole scrivere "relatore esterno".

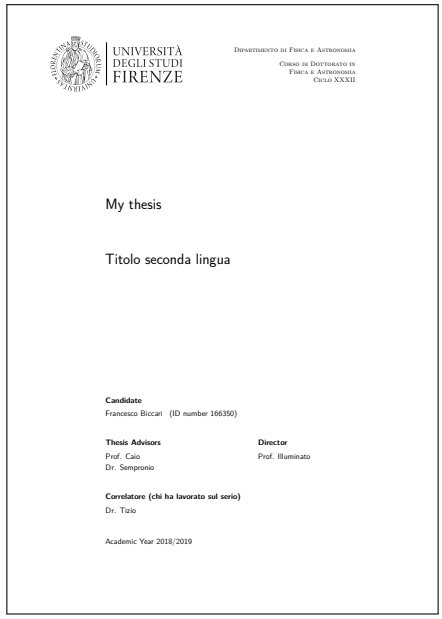

Figura 5. Esempio di frontespizio ottenuto con la classe UniFiTh usando tutti i possibili comandi di composizione del frontespizio.

`\director{...}` Opzionale. Direttore del programma di dottorato o Direttore del programma di Specializzazione in medicina e così via. Può essere ripetuto più volte come il comando `\advisor`.

`\customdirectorlabel{...}` Opzionale. Specifica l'etichetta sopra la lista dei direttori nel caso in cui la scritta di default, "Direttore", non sia adatta.

Come esempio riportiamo il seguente estratto di codice che usa tutti i possibili comandi per la composizione del frontespizio. Usandolo all'interno dell'esempio minimale riportato all'inizio dell'articolo si ottiene il frontespizio della figura 5.

```
\title{My thesis}
\alttitle{Titolo seconda lingua}
\author{Francesco Biccari}
\IDnumber{166350}
\course{Corso di Dottorato in\\
  Fisica e Astronomia}
\courseorganizer{Dipartimento di
  Fisica e Astronomia}
\cycle{Ciclo XXXII}
\advisor{Prof. Caio}
\advisor{Dr. Sempronio}
\coadvisor{Dr. Tizio}
\customcoadvisorlabel{Correlatore (chi
```

```
    ha lavorato sul serio)}
  \director{Prof. Illuminato}
  \customdirectorlabel{Director}
  \AcademicYear{2018/2019}
```

2.4. I comandi per il retro del frontespizio

Il retro del frontespizio è generato automaticamente insieme al frontespizio quando si dà il comando `\maketitle`. La composizione del retro del frontespizio si avvale di una serie di comandi, la maggior parte dei quali opzionali o utili solo per alcuni tipi di tesi. Vediamo quali sono.

`\copyyear{...}` Obbligatorio. Anno per il copyright (tipicamente l'anno in cui si discute la tesi).

`\authoremail{...}` Obbligatorio. Email dell'autore della tesi. Diventa automaticamente un link clickabile se si usa il pacchetto hyperref.

`\examdate{...}` Data dell'esame finale.
Esempio: `\examdate{16 February 2010}`.

`\examiner[...]{...}` Specifica i membri della commissione d'esame. Usare una volta per ogni membro, così come si fa per il comando `\advisor`. L'argomento opzionale serve a specificare un testo da mettere accanto al nome. Il primo nome avrà accanto, tra parentesi, la dicitura "Presidente".

`\thesistype{...}` Tipo di tesi (PhD thesis, Tesi di Dottorato, Master thesis, ecc...).

`\ISBN{...}` ISBN.

`\copyrightstatement{...}` Specifica un testo da usare come dichiarazione di copyright o per specificare come è stato rilasciato il materiale della tesi.

`\versiondate{...}` Versione della tesi. Si consiglia di usare una data.

`\website{...}` Sito web della tesi o dell'autore. Diventa automaticamente un link clickabile se si usa il pacchetto hyperref.

`\reviewer{...}` Specifica i revisori (detti *referee* o *reviewer*) della tesi. Tipicamente si applica solo alle tesi di dottorato. Stesso uso del comando `\advisor`. La lista dei revisori è preceduta da un testo che va specificato con il comando `\reviewerlabel{...}`. In ogni caso è sconsigliato riportare i revisori nella tesi.

Facciamo ora due esempi usando i comandi mostrati in precedenza. Il primo estratto di codice usa solo i comandi obbligatori. Usando le seguenti righe di codice nell'esempio minimale riportato all'inizio dell'articolo si otterrà il retro di frontespizio riportato nella figura 6 nel pannello in alto.

```
\copyyear{2020}
\authoremail{pippo@pippo.com}
```

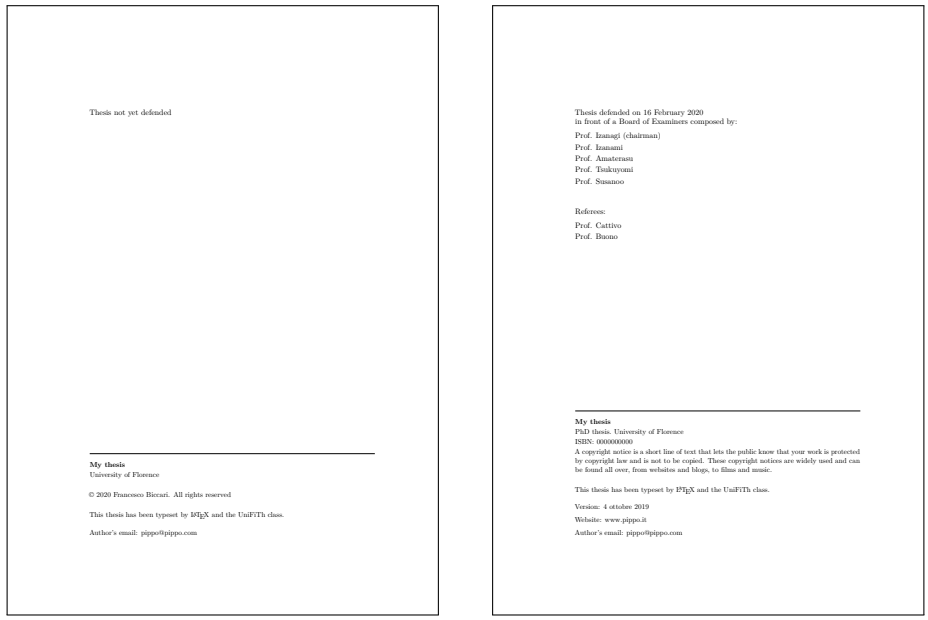

Figura 6. Esempi di retro di frontespizio ottenuti con la classe UniFiTh. A sinistra sono stati usati solo i due comandi obbligatori. A destra sono stati usati tutti i possibili comandi forniti dalla classe.

Ricordiamo che la dicitura "Tesi non ancora discussa" ("Thesis not yet defended" in inglese) può essere eliminata dando l'opzione di classe `noexaminfo` come visto sopra.

Invece, usando tutti i comandi a disposizione riportati nel seguente estratto di codice, si ottiene il retro del frontespizio riportato nella figura 6 nel pannello in basso.

```
\copyyear{2020}
\authoremail{pippo@pippo.com}
\examdate{16 February 2020}
\examiner{Prof. Izanagi}
\examiner{Prof. Izanami}
\examiner{Prof. Amaterasu}
\examiner{Prof. Tsukuyomi}
\examiner{Prof. Susanoo}
\thesistype{PhD thesis}
\ISBN{0000000000}
\copyrightstatement{A copyright notice...}
\versiondate{4 ottobre 2019}
\website{www.pippo.it}
\reviewer{Prof. Cattivo}
\reviewer{Prof. Buono}
\reviewerlabel{Referees}
```

2.5. Altri comandi per il materiale iniziale

La classe UniFiTh mette a disposizione anche i seguenti comandi per la composizione del materiale iniziale.

`\dedication{...}` Un *comando* per inserire la dedica. L'argomento è il contenuto della dedica che viene stampata in una pagina a parte.

abstract Un *ambiente* per comporre l'abstract della tesi. Ha un argomento opzionale per cambiare il titolo del paragrafo rispetto a quello di default. Questo può essere utile quando si scrive l'abstract in inglese ma la tesi in italiano, come spesso viene richiesto nei corsi di laurea scientifici. In questo caso si ricordi di dare il comando `\selectlanguage{...}` fornito dal pacchetto babel per una sillabazione corretta.

acknowledgments Un *ambiente* per comporre i ringraziamenti della tesi. Ha un argomento opzionale per cambiare il titolo del paragrafo rispetto a quello di default. Questo può essere utile quando si scrive la tesi in inglese ma i ringraziamenti in italiano. In questo caso si ricordi di dare il comando `\selectlanguage{...}` fornito dal pacchetto babel per una sillabazione corretta.

unifiblue Questo non è né un comando né un ambiente, ma è il *nome di un colore* definito appositamente per la classe UniFiTh. Corrisponde al colore RGB(0,82,147) usato nei documenti ufficiali dell'Università degli Studi di Firenze. Un esempio d'uso è `\textcolor{unifiblue}{testo da colorare...}`.

2.6. Un esempio completo

Mostriamo infine la struttura di un esempio di codice completo per la classe UniFiTh. Il risultato è riportato nelle Figure 7 e 8.

```
% righe magiche
% !TeX encoding = UTF-8
.....................

% caricamento e opzioni della classe
\documentclass[binding=0cm]{unifith}

% pacchetti utili per la tesi
\usepackage{...}
...............

% comandi per il frontespizio
\title{...}
..........

% comandi per il retro del frontespizio
\copyyear{...}
.............

\begin{document}
```

```
\frontmatter
\maketitle

\dedication{Dedicated to\\ Donald Knuth}

\begin{abstract}
.................
\end{abstract}

\begin{acknowledgments}[Ringraziamenti]
Ho deciso di scrivere i ringraziamenti in italiano...
\end{acknowledgments}

\tableofcontents

\mainmatter
\chapter{Un bel capitolo}
.........................

\chapter{Un altro bel capitolo}
.............................

\appendix
\chapter{Una bella appendice}
.......................

\backmatter
% bibliography commands

\end{document}
```

Questo esempio mostra la maniera suggerita di ordinamento delle varie parti della tesi (ringraziamenti ecc...). Inoltre nelle Figure 7 e 8 è possibile notare la geometria della gabbia del testo adottata da UniFiTh.

2.7. Comandi utili per il testo

UniFiTh mette a disposizione alcuni comandi utili per le tesi scientifiche. Le funzionalità di molti di questi comandi sono meno evolute rispetto a quelle fornite da alcuni pacchetti specifici, come per esempio nel caso di siunitx. In alcune circostanze possono però essere molto utili.

Questi comandi vengono definiti dalla classe UniFiTh subito prima del `\begin{document}` e le definizioni sono date tramite il comando `\providecommand`. In questo modo l'utente o gli altri pacchetti caricati avranno la precedenza sulla definizione di comandi con lo stesso nome di quelli forniti da UniFiTh.

`\eu` Napier's number, e, in roman.

`\iu` Imaginary unit, i, in roman.

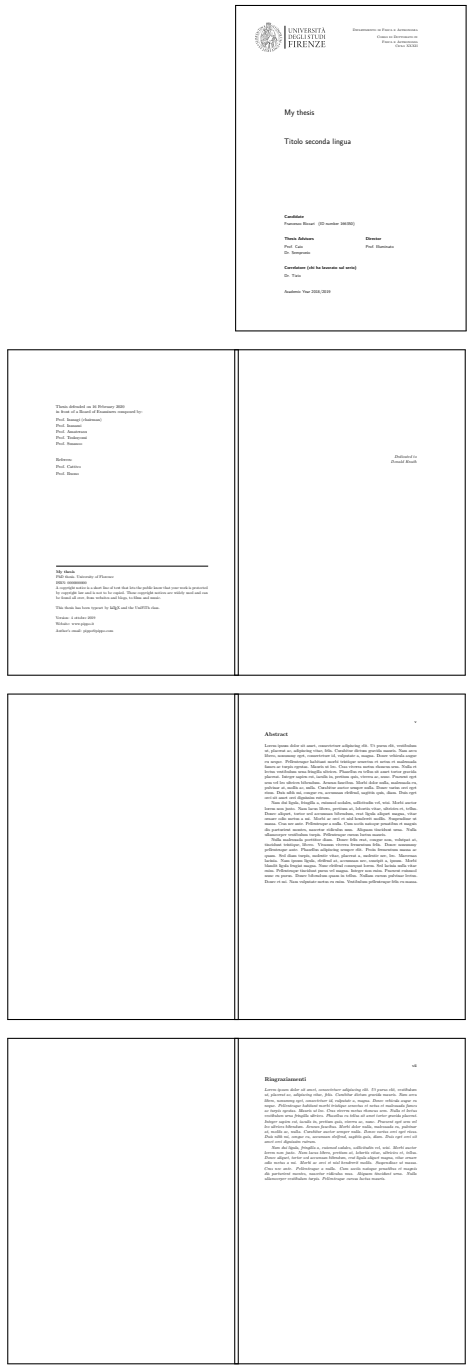

Figura 7. Le prime 7 pagine di un documento ottenuto con la classe UniFiTh. Vedi esempio completo nel testo. Le restanti pagine sono riportate nella figura 8.

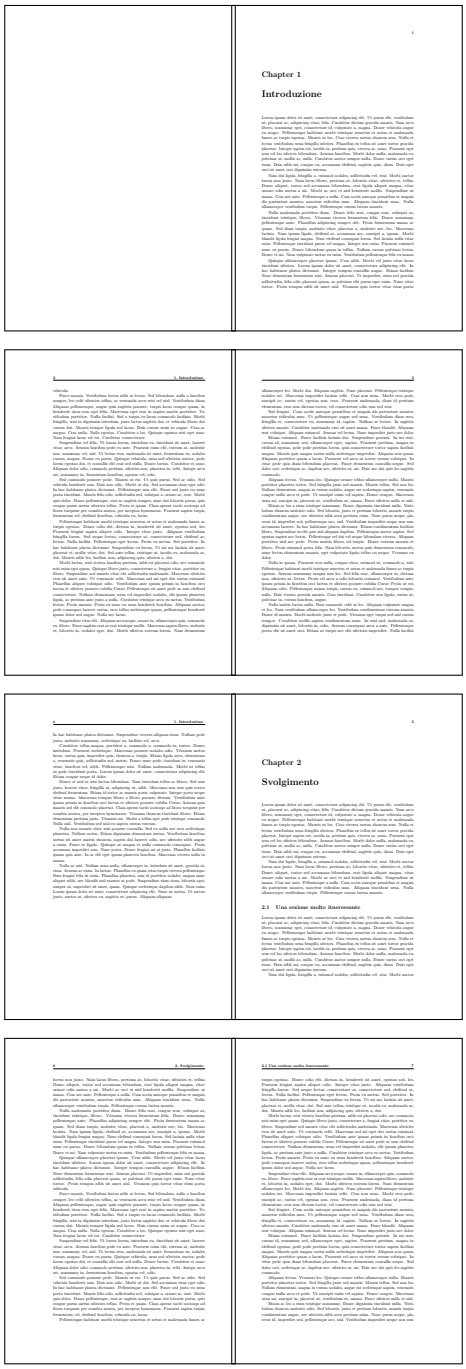

Figura 8. Pagine dalla 8 alle 15 di un documento ottenuto con la classe UniFiTh. Vedi esempio completo nel testo. Le precedenti pagine sono riportate nella figura 7.

\rb{. . . } Roman suBscript. Pedice diritto anche in ambiente matematico. Esempio:
`\[ a_b \neq a\rb{b} \]`

$$a_b \neq a_{\mathrm{b}}$$

\rp{. . . } Roman suPerscript. Apice diritto anche in ambiente matematico. Un esempio:
`\[ a^b \neq a\rp{b} \]`

$$a^b \neq a^{\mathrm{b}}$$

\tb{. . . } Text suBscript. Pedice nel testo. Un esempio:
`Cu\tb{It}: rame comprato in Italia`
$\mathrm{Cu}_{\mathrm{It}}$: rame comprato in Italia

\tp{. . . } Text suPerscript. Apice nel testo. Un esempio: `Cher G\tp{le} Napol\'eon`
Cher G$^{\mathrm{le}}$ Napoléon

\un{. . . } Comando utile per comporre le unità di misura in modo corretto, per esempio `25\un{m/s}`, `13\un{kg\,cm^{-3}}`. Può essere usato sia dentro sia fuori da un ambiente matematico. Per un uso intensivo delle unità di misura e per inserire numeri nella forma `1.4e-5`, si raccomanda l'uso del pacchetto siunitx.

\x Scorciatoia per il comando `\times`. Esempio: `$7 \x 10^5$` produce 7×10^5.

\g Scorciatoia per il comando `\degree`. Esempio: `45\g` produce 45°.

\C Scorciatoia per il comando `\celsius`. Esempio: `37\,\C` produce 37 ℃.

\A Angstrom. Esempio: `10\,\A` produce 10 Å. Si ricorda che l'angstrom non dovrebbe essere più utilizzato in testi scientifici.

\micro Prefisso micro. Esempio: `7\,\micro m` produce 7 µm. Notare che la scrittura 7 μm è sbagliata!

\ohm Ohm. Esempio: `100\,\ohm` produce 100 Ω.

\di Simbolo di differenziale con spaziatura automatica. Esempio: `$\int x \di x$` produce $\int x\,dx$. Se si preferisce che il simbolo di differenziale sia diritto (d) e non corsivo, dare l'opzione di classe `romandiff`.

\der{. . . }{. . . } Derivate. Il primo argomento rappresenta la funzione da derivare mentre il secondo le variabili rispetto alle quali derivare, separate da virgole. Il simbolo di differenziale è inserito automaticamente. L'asterisco seguito da un numero tra parentesi graffe prima della variabile indica l'ordine di derivazione. Esempi:
`\der{f}{x}`

$$\frac{df}{dx}$$

`\der{f}{x,y}`

$$\frac{d^2 f}{dx\,dy}$$

```
\der{f}{*{3}{x}}
```

$$\frac{d^3 f}{dx^3}$$

```
\der{f}{*{2}{x},*{2}{y},z}.
```

$$\frac{d^5 f}{d^2x\, d^2y\, dz}$$

`\pder{...}{...}` Derivate parziali. Stessa sintassi del comando `\der`.

`\mnote{...}` Note a margine. Si consiglia fortemente di evitare note a margine nelle tesi.

3. Analisi del codice della classe

Si discuteranno ora due parti del codice della classe UniFiTh. Sono state scelte perché potrebbero essere utili in molte altre situazioni quando si scrivono comandi personalizzati.

3.1. Il problema del logo

Come anticipato precedentemente, il logo dell'Università degli Studi di Firenze non può essere distribuito sul CTAN e nelle distribuzioni TEX. Pertanto, la prima volta che si compila, appare un avviso nel frontespizio come riportato nella figura 9:

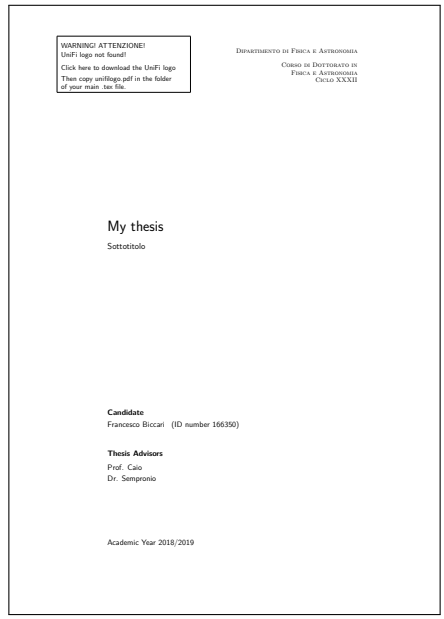

Figura 9. Esempio di frontespizio ottenuto con la classe UniFiTh nel caso in cui il logo dell'Università degli Studi di Firenze non sia presente.

Quel messaggio è stato ottenuto con due blocchi di codice. Il primo è il seguente e viene eseguito tutte le volte che viene compilato il documento.

```
\newif\ifUFI@unifilogo \UFI@unifilogofalse
\IfFileExists{unifilogo.pdf}{\UFI@unifilogotrue}{%
\ClassWarningNoLine{unifith}{Logo not found!
You have to download it manually. See the
documentation or just the frontispiece.}
\RequirePackage{hyperref}}
```

Grazie alla funzione `\IfFileExists` si controlla se il file `unifilogo.pdf` esiste. In base al risultato la variabile booleana `\UFI@unifilogo` viene impostata al valore vero o falso. Se il file non viene trovato viene caricato il pacchetto hyperref che servirà a rendere clickabile il link sul frontespizio per scaricare il logo.

Successivamente, dentro il codice del frontespizio, che viene eseguito quando si dà il comando `\maketitle`, si trova:

```
\ifUFI@unifilogo
  \parbox[b][3cm][c]{0.49\linewidth}{\%
  \hspace{-1cm}%
  \includegraphics[width=6.5cm]{unifilogo.pdf}}
\else
  \parbox[b][3cm][c]{0.49\linewidth}{\hspace{-1cm}
  \setlength{\unitlength}{1mm}
  \begin{picture}(65,27.3)(0,0)
  \small
  \put(0,0){\framebox(65,27.3)[0,0]{}}
  \put(2,23){\makebox(0,0)[l]{%
  WARNING! ATTENZIONE!}}
  \put(2,18){\makebox(0,0)[l]{%
  UniFi logo not found!}}
  \put(2,11){%
  \href{http://.../unifilogo.pdf}%
  {Click here to download the UniFi logo}}
  \put(2,5.5){Then copy%
  unifilogo.pdf in the folder}
  \put(2,1.5){of your main .tex file.}
  \end{picture}
  }
\fi
```

In questo caso, semplicemente, in base alla presenza o meno del file del logo (`\UFI@unifilogo`), viene caricato il logo oppure viene disegnato il rettangolo con l'avvertimento e il link grazie all'ambiente picture di LaTeX.

3.2. Il comando `\advisor`

Un'altra parte di codice interessante da analizzare è il comando `\advisor`. Allo stesso modo funzionano i comandi `\coadvisor`, `\director`, ecc…

```
\newcount\UFI@advisorcount
\newtoks\UFI@advisortoks
\newcommand{\advisor}[1]{%
  \ifnum\UFI@advisorcount=\z@
    \UFI@advisortoks={#1}%
  \else
    \UFI@advisortoks=\expandafter{%
    \the\UFI@advisortoks\\[0.75mm]#1}%
  \fi
  \advance\UFI@advisorcount\@ne}
```

Nel codice vengono definiti un contatore (`\UFI@advisorcount`) e un array di token, (`\UFI@advisortoks`) cioè in pratica una stringa di testo. Quando `\advisor` viene eseguito, si controlla prima di tutto se è la prima volta (contatore a zero). In quel caso si assegna il suo argomento, cioè il nome del relatore, alla stringa di testo e si incrementa il contatore di una unità.

Durante le esecuzioni successive (cioè nel caso in cui chi scrive la tesi abbia dato più volte questo comando), si finisce nel secondo ramo dell'`ifnum`. In quel caso viene aggiornato il contenuto della stringa `\UFI@advisortoks` inserendo quella vecchia, un a capo, e infine il nuovo nome contenuto nell'argomento del comando. Tutto questo è reso possibile grazie all'uso di `\expandafter`. Questo comando semplicemente ritarda l'espansione del token immediatamente successivo, in questo caso la parentesi graffa, che però non è espandibile. Serve quindi semplicemente a ritardare l'esecuzione dell'assegnazione del contenuto delle graffe alla stringa. Per esempio supponiamo che questo sia il codice nel sorgente della tesi

```
\advisor{Pippo}
\advisor{Pluto}
```

Durante l'esecuzione del secondo comando `\advisor` l'assegnazione non viene eseguita subito, perché c'è un `\expandafter`. Dopo il primo livello di espansione si ottiene

```
UFI@advisortoks={Pippo\\[0.75mm]Pluto}
```

Solo ora, al secondo livello di espansione, alla stringa verrà assegnato il nuovo valore contenuto tra parentesi graffe.

4. Due problemi tipici nella preparazione delle tesi

Tratteremo ora due argomenti che gli studenti incontrano spesso durante la stesura delle tesi. Il primo riguarda la dimensione del file della tesi. L'uso di immagini non ottimizzate o malamente convertite porta sovente a dimensioni del pdf di decine e decine, se non centinaia, di MB. Come ridurre queste dimensioni? Il secondo problema è relativo alla preparazione di PDF archiviabili (PDF/A), cioè file PDF che devono aderire a specifiche molto stringenti per la loro visualizzazione anche a distanza di anni. Molte università iniziano a richiedere che i documenti consegnati siano PDF/A. Ma come si creano?

4.1. Ridurre la dimensione del file della tesi

Per ridurre le dimensioni di un pdf troppo grande, supponendo che il problema derivi principalmente dalle immagini del pdf, si può procedere lungo due strade. La prima è quella di riottimizzare tutte le immagini. Ma pochi studenti sanno farlo e spesso non hanno neanche il tempo di farlo. L'altra strada consiste nel ricodificare un pdf.

Usando Ghostscript (AA.VV., a) possiamo *distillare* un file pdf, cioè interpretarlo e ricodificarlo cambiando le impostazioni. Ecco una lista di possibili comandi che si possono lanciare dal terminale (i comandi sono intesi per Ghostscript su Windows 64 bit ma possono essere adattati velocemente a un altro sistema operativo cambiando il nome dell'eseguibile di Ghostscript).

Nel caso in cui si voglia una compressione delle immagini di tipo DECT, cioè con perdita (in pratica jpeg), si può usare il comando

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

L'opzione `/prepress` indica la migliore qualità possibile. Se però si vuole ottenere una dimensione del file ancora minore si può sostituire `/prepress` con `/printer`, `/ebook`, oppure `/screen` (in ordine di dimensione decrescente, così come la qualità delle immagini risultanti).

Nel caso in cui si voglia una compressione delle immagini di tipo Flate, cioè senza perdita, si può usare il comando

```
gswin64c.exe -q -dNOPAUSE -dBATCH -dSAFER
-sDEVICE=pdfwrite -sOutputFile=output.pdf
-dAutoFilterColorImages=false
-dAutoFilterGrayImages=false
-dColorImageFilter=/FlateEncode
-dGrayImageFilter=/FlateEncode
-dPDFSETTINGS=/prepress
-dCompatibilityLevel=1.5 input.pdf
```

4.2. Creare un pdf archiviabile

Il PDF/A (Wikipedia, 2021) è una particolare specifica del formato PDF, standardizzata dall'ISO, pensata per archiviare e preservare documenti elettronici per lungo tempo. Istituzioni pubbliche e private, tra le quali molte università, chiedono e usano sempre più questo formato per archiviare documenti elettronici. Ci sono diversi livelli delle specifiche PDF/A. Il formato suggerito per le tesi è il PDF/A-2b.

La maniera migliore e gratuita di controllare se un pdf aderisce correttamente alle specifiche di uno dei livelli del PDF/A è quella di usare veraPDF (AA.VV., d), un software open source supportato dalle più importanti associazioni del mondo legate al formato PDF e finanziato dalla Commissione Europea. È liberamente disponibile per Windows, Linux, e Mac.

Un file PDF/A può essere generato direttamente in LaTeX, come spiegato da Claudio Beccari (Beccari, 2020). Quello che si mostrerà in questo articolo è invece la conversione di un normale pdf ottenuto dalla compilazione con `pdflatex` in un PDF/A. Questa situazione

infatti potrebbe essere più veloce per uno studente. Useremo, come al solito, Ghostscript (AA.VV., a).

Ecco i passi da seguire:

1. Copiare nella propria cartella di lavoro un profilo di colore `icc`. Dato che il pdf va visto su schermo, va scaricato un profilo di colore di tipo sRGB. Uno generico va più che bene. Può essere scaricato dal sito ufficiale `https://www.color.org/srgbprofiles.xalter`. Supponiamo di rinominarlo `srgb.icc`

2. Copiare un file di definizioni Postscript adatto alla conversione in PDF/A nella propria cartella di lavoro. Questo file, praticamente già quasi pronto, può essere copiato dal vostro stesso computer. Lo trovate sotto il nome `PDFA_def.ps` nella cartella `lib` della vostra installazione di Ghostscript (per esempio `C:\Program Files\gs\gs9.22\lib\`). Copiate questo file nella vostra cartella di lavoro e soprattutto rinominatelo! Supponiamo di rinominarlo `PDFArenamed.ps`

3. Modificare opportunamente il file di definizioni Postscript. In pratica dovete aprire il file `PDFArenamed.ps` e specificare al suo interno il profilo di colore (`srgb.icc`) e il titolo del vostro file pdf. Per farlo, usate un editor di testo qualsiasi. Le variabili da modificare sono intuitive e facilmente identificabili: `/ICCProfile` e `/Title`.

4. Aprire un terminale e lanciare il seguente comando:

```
gswin64.exe -q -dNOPAUSE -dBATCH
-dNOSAFER -sDEVICE=pdfwrite
-sOutputFile=output.pdf
-dPDFA=2 -dPDFACompatibilityPolicy=1
-sColorConversionStrategy=UseDeviceIndependentColor
-sProcessColorModel=DeviceRGB
PDFArenamed.ps input.pdf
```

5. Infine si consiglia di controllare che il file ottenuto dalla conversione aderisca effettivamente alle specifiche PDF/A con il software VeraPDF.

5. Conclusioni e prospettive

In questo articolo è stata presentata la classe UniFiTh, una classe LaTeX per la composizione delle tesi dell'Università degli Studi di Firenze.

Nelle future versioni si prevede di aggiornare soprattutto la documentazione, trattando quei casi d'uso più tipici o complessi. Per esempio, si prevede di aggiungere un paragrafo con alcuni consigli e stili sulla bibliografia, un paragrafo sull'uso di UniFiTh per tesi di carattere umanistico, e infine un paragrafo sull'uso di Overleaf (AA.VV., b), uno strumento sempre più in uso in tutte le università.

Riferimenti bibliografici

AA.VV. (a). «Ghostscript software». URL https://www.ghostscript.com/. Controllato il 25 ottobre 2021.

— (b). «Overleaf». URL https://www.overleaf.com/. Controllato il 25 ottobre 2021.

— (c). «Piano di comunicazione dell'Università degli Studi di Firenze». URL https://www.unifi.it/ls-36-unifi-comunica.html. Controllato il 25 ottobre 2021.

— (d). «VeraPDF». URL https://verapdf.org/. Controllato il 25 ottobre 2021.

Beccari, C. (2020). «Creare file archiviabili con pdfLaTeX e lualaTeX». URL http://www.guitex.org/home/images/doc/GuideGuIT/FileArchiviabili.pdf. Controllato il 25 ottobre 2021.

Biccari, F. (2019). «UniFiTh: una classe per la composizione delle tesi dell'Università di Firenze». Forum del GuIt. URL https://www.guitex.org/home/it/forum/5-tex-e-latex/115954. 23 aprile 2019.

— (2021). «The UniFiTh package». URL https://ctan.org/pkg/unifith. Controllato il 25 ottobre 2021.

Wikipedia (2021). «PDF/A». URL https://it.wikipedia.org/wiki/PDF/A. Controllato il 25 ottobre 2021.

Francesco Biccari
Dipartimento di Fisica e Astronomia
Università degli Studi di Firenze
Firenze, Italia
biccari@gmail.com

Personalizzare TEXworks

Filippo Vomiero

Sommario TEXworks è l'editor incluso nelle installazioni di TEX Live per Windows e MiKTEX: a prima vista può sembrare molto semplice, soprattutto se comparato con i più complessi TeXstudio o TeXmaker. Tuttavia, TEXworks è anche estremamente personalizzabile e questo lo rende versatile e facilmente adattabile alle proprie esigenze. Questo articolo illustra come personalizzare alcuni aspetti di TEXworks: il tema dell'interfaccia, la colorazione della sintassi, i dizionari e gli strumenti di composizione.

Abstract TEXworks is the default editor bundled with TEX Live for Windows and MiKTEX: at first glance it seems a quite minimalist program, even more so if compared to the more feature-rich TeXstudio or TeXmaker. TEXworks, however, is also highly customizable, so it is more suitable to be crafted fitting one's own needs. The article will show a few methods to customize TEXworks features like interface theme, syntax highlighting, dictionaries and processing tools.

1. Introduzione

Per alcuni anni, come passatempo, mi sono divertito a decompilare applicazioni Android per modificare la combinazione di colori predefinita, soprattutto a partire da Android 5 con l'introduzione del *Material design* che ha reso tutte le applicazioni a sfondo bianco, scelta che ho sempre trovato molto fastidiosa per gli occhi, soprattutto se si legge di sera con poca luce. Passando al mondo LATEX, è stato per me naturale provare a modificare anche TEXworks, il programma incluso nelle installazioni di TEX Live per Windows e MiKTEX.[1]

TEXworks è un programma open source, multi-piattaforma, sviluppato con la libreria Qt, ispirato a TeXShop, l'attuale editor predefinito in MacTeX. Nella mia ricerca ho constatato che il programma è ampiamente configurabile, sia per quanto riguarda le funzionalità, sia nell'estetica dell'interfaccia: l'articolo mostrerà sul piano delle funzioni come aggiungere e configurare dizionari e strumenti di composizione, mentre sul piano dell'estetica come modificare il tema dell'interfaccia e la colorazione della sintassi.

Prima di continuare, una premessa importante: TEXworks è multi-piattaforma e viene rilasciato per Windows, per le principali distribuzioni Linux e per macOS[2]; nell'articolo, tuttavia, tutti gli esempi si riferiscono a Windows. Nella maggior parte dei casi il cambio di sistema operativo non dovrebbe costituire un problema, visto che sono coinvolti linguaggi svincolati dalla piattaforma (*regex* e *Qt*), ma molto probabilmente le parti che riguardano i file di configurazione sono valide solo in Windows, e necessitano di qualche adattamento per essere applicate negli altri sistemi operativi. La questione va oltre le mie capacità, ma la

1. Esistono programmi molto validi come TeXstudio e TeXmaker che supportano nativamente i temi e danno la possibilità di impostare un tema scuro. TEXworks, tuttavia, è un programma ben più leggero e rapido nell'esecuzione, oltre a garantire una maggior libertà di configurazione.
2. In GitHub sono disponibili anche i sorgenti che possono essere compilati per altre piattaforme.

comunità del GUIT è ricca di esperti in grado di risolvere questi piccoli problemi, e il forum esiste anche per questo.

2. La cartella delle risorse

Il primo passo da compiere per personalizzare TeXworks è capire dove sono memorizzati i file di configurazione. Nel menu `aiuto` di TeXworks, è presente la voce 'preferenze e risorse', che permette di individuare dove sono memorizzate tali informazioni. Le preferenze sono salvate, di norma, in un file `texworks.ini`, o, se stiamo lavorando in Windows, possono essere salvate nel registro di configurazione di sistema. Le preferenze si possono modificare direttamente da GUI, e vengono salvate in automatico. Le risorse invece sono memorizzate in una cartella dedicata che contiene vari file di testo corrispondenti a specifiche funzioni: per personalizzare l'installazione di TeXworks vanno modificati manualmente, e nelle prossime sezioni vedremo come fare.

La cartella delle risorse contiene al suo interno queste sotto-cartelle:

completion Contiene i file con tutte le voci per la funzione di auto-completamento
configuration Cartella con file di configurazione per varie funzioni:

- rientro automatico
- coppie di delimitatori
- virgolette intelligenti
- colorazione della sintassi
- *tag* di sezione (visibili dal menu Finestra, visualizza, tags)
- configurazione avanzata, per delle impostazioni non modificabili dal menu preferenze

dictionaries Contenitore per i dizionari disponibili per il controllo ortografico
scripts TeXworks permette di eseguire degli script per svolgere funzioni aggiuntive, nella cartella sono presenti alcuni esempi
translations Cartella per testare localmente una traduzione dell'interfaccia
TUG Posizione di default del file `texworks.ini`

In ambiente Windows e con TeX Live, di default viene creata una cartella, il cui nome corrisponde alla versione di TeX Live in uso, dentro alla cartella principale dell'account utente, ad esempio:

`C:\Users\`⟨*nome utente*⟩`\.texlive2021`

Le preferenze e le risorse di TeXworks vengono salvate in un'ulteriore sotto-cartella `configuration`. Ad ogni cambio di versione si rende quindi necessario trasferire tutti i file personalizzati nella cartella della nuova versione (ad esempio `.texlive2022`). Ci sono due possibili strade per risolvere il problema. La prima consiste nel creare un file `texworks-setup.ini` nella cartella con l'eseguibile di TeXworks.[3] All'interno del file vanno indicati i percorsi da utilizzare per salvare i file di configurazione:

3. In Windows va usata la cartella:
⟨*TeX Live*⟩`\tlpkg\texworks\texworks.exe`

```
inipath=C:/myfolder/TW_conf/
libpath=C:/myfolder/TW_conf/
```

La prima voce, `inipath`, indica la posizione del file `texworks.ini`, mentre `libpath` indica la posizione della cartella delle risorse. L'esempio qui riportato, ripreso dal manuale TEXworks (Delmotte *et al.*, 2021, 29–30), usa `TW_conf`; ciascuno può scegliere il nome che preferisce, ma la cartella non verrà creata in automatico. Si noti, inoltre, l'uso della barra dritta al posto di quella rovescia solitamente usata in ambiente Windows. Se si usa TEX Live, può essere una scelta efficace usare la cartella personale `texmf` prevista dalla *TEX Directory Structure* (TDS).[4]

La seconda soluzione, valida solo in ambiente Windows, consiste nel modificare il collegamento che si usa per aprire TEXworks e usare esclusivamente quello (può apparire scomodo, ma è necessario anche per usare un tema personalizzato, come si vedrà nella sezione 5). Il collegamento all'eseguibile di TEXworks creato durante l'installazione di TEX Live punta a:

⟨*TEX Live*⟩`\bin\win32\texworks.exe`

Se si modifica in:

⟨*TEX Live*⟩`\tlpkg\texworks\texworks.exe`

le impostazioni verranno salvate nel registro di configurazione, mentre la cartella delle risorse verrà salvata in:

`C:\Users\`⟨*utente*⟩`\AppData\Roaming\TUG\TeXworks`

risultando così immune ai vari cambi di versione annuali.

3. I dizionari

Una volta individuata la posizione della cartella di configurazione del programma, possiamo iniziare con le personalizzazioni. La più semplice è quella di aggiungere la cartella con i file dei dizionari, così da poter usufruire del controllo ortografico durante la digitazione (cfr. Gregorio, 2010, 12–13). Il controllo è effettuato dal programma Hunspell, lo stesso usato da Open/Libre Office, Firefox e Thunderbird. Per usufruire di un dizionario è sufficiente creare la cartella `dictionaries` dentro alla cartella delle risorse; al suo interno inseriremo i dizionari delle lingue in cui siamo soliti scrivere.[5] È possibile recuperare i file dei dizionari dai repository di estensioni di OpenOffice o LibreOffice:
`https://extensions.openoffice.org/`
`https://extensions.libreoffice.org/`

Una volta fatto, entriamo nella finestra 'preferenze' di TEXworks (menu `modifica`, preferenze), selezioniamo la scheda 'editor' e nel menu a tendina corrispondente alla voce 'lingua verifica ortografia' potremo selezionare la lingua di uno dei dizionari installati.

TEXworks viene anche distribuito con uno script automatico in grado di rilevare la lingua di un documento indicata nelle opzioni di babel e impostare la verifica dell'ortografia di

4. L'esatta posizione di `texmf` cambia a seconda del sistema operativo: in Windows si trova in `%USERPROFILE%\texmf`, in macOS e nei sistemi Linux di norma è in `$HOME/texmf`.
5. La procedura è valida per Windows e macOS; nei sistemi GNU/Linux invece i dizionari sono in una cartella condivisa da tutti i programmi che ne fanno uso; per questo motivo è possibile che i dizionari richiesti siano già installati. Il percorso è `/usr/share/myspell/dicts`.

conseguenza. Lo script, tuttavia, non è configurato per la lingua italiana, per cui è necessaria una piccola modifica. Il file dello script si chiama `babelLanguage.js` (cartella `scripts/Hooks`), al cui interno va aggiunta la seguente stringa alla lista di lingue supportate:

```
spellingDict.italian = “it_IT”;
```

4. Strumenti di composizione

Un altro tipo di personalizzazione sul piano delle funzionalità riguarda la possibilità di aggiungere o modificare gli strumenti di composizione disponibili in TeXworks e accessibili tramite il menu a tendina in alto a sinistra nell'interfaccia. I principali strumenti di composizione sono già supportati nativamente, ma nella finestra preferenze, alla scheda composizione, è possibile aggiungere, rimuovere o modificare gli strumenti, oltre a cambiare l'ordine in cui appaiono nel menu a tendina e stabilire lo strumento di default.

Nella mia installazione ho aggiunto, per i tre motori principali (pdfLaTeX, LuaLaTeX e XeLaTeX), una versione alternativa con l'argomento `-shell-escape`, che permette al programma di lanciare programmi esterni, come gregorio o abcm2ps per la scrittura di musica con LaTeX. La procedura è molto semplice: dalla scheda composizione nelle preferenze, si preme il pulsante “+” per creare un nuovo strumento e si procede con la compilazione dei campi. Vediamo un esempio con pdfLaTeX:

```
Nome: pdfLaTeX+se
Programma: pdflatex.exe
```

Gli argomenti si aggiungono usando il pulsante “+” sulla destra, ne servono tre:

```
--shell-escape
$synctexoption
$fullname
```

Il primo argomento è ovvio, il secondo serve a passare al programma di compilazione l'opzione `-synctex=1` per permettere la sincronizzazione tra il file sorgente e il `pdf`. Il terzo argomento passa al programma il nome completo di estensione del file da cui avviamo la compilazione.

Ulteriori informazioni si trovano sulla pagina GitHub di TeXworks:

```
https://github.com/TeXworks/texworks/wiki/AdvancedTypesettingTools
```

4.1. Configurazioni aggiuntive

Se avete impostato TeXworks per utilizzare anche altri strumenti di composizione, potreste desiderare che i file prodotti da quegli strumenti (con le relative estensioni) vengano riconosciuti dal programma nella finestra di apertura dei file. Per ottenere questo, bisogna modificare il file `texworks-config.txt` che si trova nella cartella `configuration`. Una volta aperto il file, verso la fine si vedono una serie di righe commentate che iniziano con `# file-open-filter:`. Le righe vanno de-commentate per abilitare la personalizzazione dei tipi di file supportati (come spiegato nel file stesso). A quel punto, si potranno inserire nuove righe con le definizioni dei file che vogliamo implementare.[6] Ad esempio, per i file usati da GregorioTeX, va aggiunta

6. Esula dallo scopo di questa guida approfondire, ma va segnalato che il programma è così versatile che inserendo la definizione relativa a un tipo specifico di file e costruendo un relativo set di regole per la colorazione della sintassi

una riga in questo modo:

```
file-open-filter: Gabc score (*.gabc)
```

Sempre nello stesso file è possibile anche impostare degli schemi per la pulizia dei file ausiliari: per i programmi principali ci sono già le definizioni necessarie, ma se gli strumenti di composizione aggiunti producono a loro volta dei file ausiliari con estensione specifica, è possibile includerli negli schemi per la pulizia. L'operazione è spiegata bene, con tanto di esempio, nella pagina di documentazione di GregorioTeX relativa a TEXworks:

https://gregorio-project.github.io/configuration-texworks.html

5. Configurare un tema scuro

TEXworks non permette, per ora, di impostare un tema per l'interfaccia; tuttavia, il toolkit Qt con cui è compilato, permette di caricare un foglio di stile esterno con cui personalizzare l'aspetto dell'applicazione.[7]

Il primo passo è creare un file `css`, da salvare in una cartella dedicata; quando si lancia TEXworks va usata una opzione specifica per fargli caricare il foglio di stile (il percorso è quello mostrato nella sezione 2, valido in ambiente Windows):

"⟨*TEXworks*⟩`\texworks.exe" -stylesheet "`⟨*cartella foglio di stile*⟩`\`⟨*nome file*⟩`.css"`

Se digitato direttamente nel prompt dei comandi, non vanno usate le virgolette; la soluzione più pratica è quella di creare un collegamento aggiungendo alla chiamata del programma l'argomento appena visto.

5.1. Documentazione

La parte più impegnativa è costruire correttamente il foglio di stile; la soluzione illustrata nella sezione 5.2 serve a creare un tema a sfondo nero (visibile nella figura 1) ed è stata ottenuta dopo molti tentativi ed esperimenti. In rete non esiste molta documentazione specifica; io sono partito da una discussione su Stack Exchange (2017) generata dalla domanda "come posso ottenere un tema scuro su TEXworks?": la soluzione proposta da Robert Zhang mi ha dato lo spunto iniziale, e ho capito che con un po' di pazienza si poteva ottenere un buon risultato.

La seconda lettura obbligata è stata la ricca documentazione di Qt, che però va adattata caso per caso a ciò che è pertinente con l'interfaccia di TEXworks. Riporto qui di seguito le tre pagine più utili:

- https://doc.qt.io/qt-5/stylesheet-reference.html
- https://doc.qt.io/qt-5/stylesheet-examples.html
- https://doc.qt.io/qt-5/stylesheet-customizing.html

Da queste pagine si possono trovare ulteriori link per approfondire aspetti specifici.

(vedi la sezione 6), TEXworks idealmente può essere usato come editor per qualsiasi tipo di file, non necessariamente legati al mondo LATEX(come gli script in Javascript o i fogli di stile `css` per citare due tipi nominati in questo testo). Il sito sugli script di TEXworks mantenuto da Paul A. Norman illustra come configurare il programma per l'editing di file di script Qt: `http://twscript.paulanorman.info`

7. Al momento della scrittura di questo articolo, l'ultima versione di TEXworks è la `0.6.6`. Gli sviluppatori non escludono un futuro supporto nativo ai temi, ma non è possibile dire quando questo avverrà.

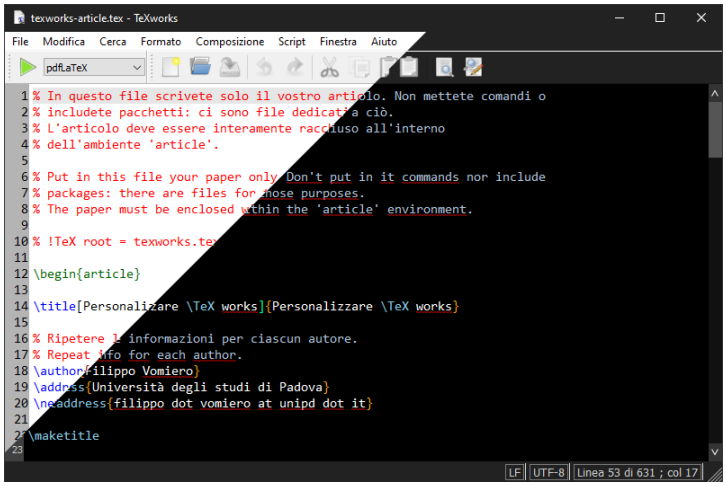

Figura 1. Tema chiaro e scuro a confronto.

5.2. Il foglio di stile

Un foglio di stile è un elenco di varie definizioni, così composte: la dichiarazione del nome dell'elemento che si vuole personalizzare e, tra graffe, le proprietà che si vogliono modificare (come, ad esempio, il colore di sfondo) con il valore da assegnare (nel caso di un colore il nome del colore o il suo valore `rgb`).

Ecco la prima definizione:

Definizione 1. QWidget

```
QWidget {
    background-color: #222222;
    selection-background-color: #555555;
    color: #cccccc;
    selection-color: white;
}
```

`QWidget` è l'elemento principale che stabilisce la combinazione di colori che verrà usata prevalentemente in tutta l'interfaccia di TEXworks. Le proprietà modificate sono il colore di sfondo e il colore in primo piano (usato per il testo). Si può notare come ci siano anche le definizioni per quando l'elemento è selezionato: queste proprietà relative allo stato sono importanti per ottenere un'interfaccia dinamica, che sembra rispondere alle azioni dell'utente.

Definizione 2. QTextEdit

```
QTextEdit {
    background-color: #000000;
    color: white;
    selection-background-color: #555555;
    selection-color: white;
}
```

Questa è la finestra dell'editor di testo. Le proprietà modificate sono le stesse di `QWidget`. In questo modo l'aspetto di TEXworks apparirà con una cornice grigio scuro, mentre la sezione principale dove si scrive sarà nera con testo bianco (oltre alla colorazione in base alla sintassi stabilita nella sezione precedente). Ho inserito anche uno sfondo diverso per il testo selezionato, come visibile nella figura 2.

Definizione 3. QComboBox

```
QComboBox {
    border: 1px solid gray;
    padding: 2px 18px 4px 6px;
    border-radius: 4px;
}
QComboBox::drop-down {
    subcontrol-origin: padding;
    subcontrol-position: top right;
    width: 20px;
    border-left-width: 1px;
    border-left-color: #eeeeee;
    border-left-style: solid;
}
QComboBox QAbstractItemView {
    border: 2px solid darkgray;
    selection-background-color: #333333;
    background-color: black;
}
QComboBox::down-arrow {
    image: url(./res/downarrow.png);
    width: 10px;
    height: 10px;
}
QComboBox::down-arrow:on {
    top: 1px;
    left: 1px;
}
```

`QComboBox` è l'elemento che riguarda i menu a tendina, come quello per scegliere il motore di compilazione in alto a sinistra nella finestra di TEXworks, visibile nella figura 3. La prima definizione riguarda la parte dove è presente il testo (per esempio pdfLaTeX), la seconda invece il riquadro della freccia. La terza definisce l'aspetto del menu una volta aperto, la quarta indica un'immagine per la freccia e le sue dimensioni, e la quinta la sposta a destra e in basso di un pixel (lo spostamento è ottenuto impostando un pixel di margine in alto e a sinistra) per dare l'impressione che il pulsante sia stato premuto.

Come si può notare, l'immagine per la freccia fa riferimento a una risorsa esterna, un'immagine png che va a rimpiazzare quella originale, la quale, essendo pensata per uno sfondo chiaro, è completamente nera e quindi poco visibile. L'immagine a cui si fa riferimento è una piccola freccia che ho disegnato con Inkscape e convertito in png. Conviene inserire tutte le immagini che andremo a utilizzare in un'unica cartella, nell'esempio io l'ho chiamata `res` (per *resources*), da posizionare nella stessa cartella dove abbiamo salvato il file `css`.

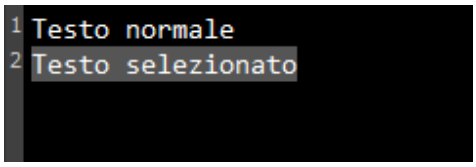

Figura 2. QTextEdit

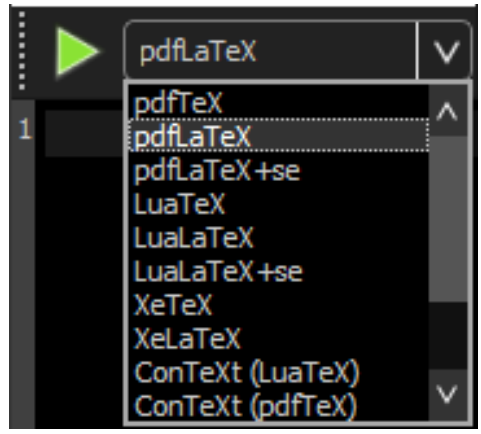

Figura 3. QComboBox

Definizione 4. QMenu

```
QMenu::item:selected {
    background-color: #333333;
}

QMenuBar::item:pressed, QMenuBar::item:selected {
    background: #333333;
}
```

Ho raggruppato qui due elementi che riguardano i menu principali: `QMenuBar` definisce l'aspetto dei pulsanti del menu principale (file, modifica, cerca, ecc.), mentre `QMenu::item` permette di definire l'aspetto delle voci di un menu una volta aperto. In entrambi i casi ho impostato un colore leggermente più chiaro quando una voce è selezionata (figura 4).

Definizione 5. QScrollBar:vertical

```
QScrollBar:vertical {
    background: #111111;
    width: 15px;
    margin: 20px 0 20px 0;
}
QScrollBar::handle:vertical {
    background: #555555;
    min-height: 20px;
}
QScrollBar::add-line:vertical {
     border: none;
     background: #333333;
     height: 20px;
     subcontrol-position: bottom;
     subcontrol-origin: margin;
}
```

```
QScrollBar::sub-line:vertical {
      border: none;
      background: #333333;
      height: 20px;
      subcontrol-position: top;
      subcontrol-origin: margin;
}
QScrollBar::add-line:vertical:pressed {
    background: #444444;
}
QScrollBar::sub-line:vertical:pressed {
    background: #444444;
}
QScrollBar::up-arrow:vertical {
    image: url(./res/uparrow.png);
    border: none;
    width: 8px;
    height: 8px;
    padding: 12px;
}
QScrollBar::down-arrow:vertical {
    image: url(./res/downarrow.png);
    border: none;
    width: 8px;
    height: 8px;
    padding: 12px;
}
QScrollBar::add-page:vertical, QScrollBar::sub-page:vertical {
   background: none;
}
```

`QScrollBar`, come dice il nome, riguarda le barre di scorrimento (figura 5), in questo caso quelle verticali, ma la definizione andrà replicata anche per quelle orizzontali. La prima definizione stabilisce il colore di sfondo della barra (più scuro rispetto al resto dell'interfaccia), la larghezza della barra e i margini: vengono lasciati 20 pixel di margine sopra e sotto, che è lo spazio che serve per disegnare i due pulsanti con le frecce. `Handle` è la parte in movimento della barra, qui disegnata più chiara; viene inoltre stabilita una grandezza (altezza) minima.

La terza e la quarta definizione creano due piccole barre sopra e sotto più chiare, che fungono da pulsante, e contengono le frecce. Vale la pena spiegare un paio di proprietà: `subcontrol-origin` indica da quale punto si stabilisce la posizione dell'elemento, in questo caso la linea del margine (che è il rettangolo più esterno), mentre `subcontrol-position` indica quale margine considerare, tra le quattro direzioni. Le due definizioni che seguono servono a rendere più chiaro il pulsante quando viene premuto.

È importante ricordare che quando si decide di modificare lo stile di un elemento, come in questo caso `QScrollBar`, la sintassi di Qt vuole che vengano definite tutte le sue parti: se non si indica un'opzione per gli elementi grafici, in questo caso le frecce, essi non verranno disegnati e al loro posto appariranno due piccoli quadrati. Si spiegano così le due definizioni successive. Oltre all'immagine da usare, viene eliminato il bordo, stabilite le dimensioni e il *padding*, cioè la distanza interna tra bordo e contenuto.

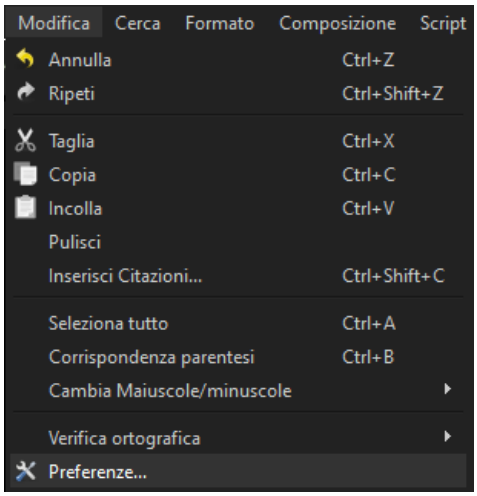

Figura 4. QMenu e QMenuBar.

Figura 5. QScrollBar

L'ultima definizione stabilisce l'aspetto dello sfondo della barra sopra e sotto alla parte in movimento: con none è a tinta unita, senza questa indicazione apparirebbe a scacchi.

Definizione 6. QScrollBar:horizontal

```
QScrollBar:horizontal {
    background: #111111;
    height: 15px;
    margin: 0 20px 0 20px;
}
QScrollBar::handle:horizontal {
    background: #555555;
    min-width: 20px;
}
QScrollBar::add-line:horizontal {
     border: none;
     background: #333333;
    width: 20px;
    subcontrol-position: right;
    subcontrol-origin: margin;
}
QScrollBar::sub-line:horizontal {
     border: none;
     background: #333333;
    width: 20px;
    subcontrol-position: left;
    subcontrol-origin: margin;
}
QScrollBar::add-line:horizontal:pressed {
    background: #444444;
```

```
}
QScrollBar::sub-line:horizontal:pressed {
    background: #444444;
}
QScrollBar:left-arrow:horizontal {
    image: url(./res/leftarrow.png);
    border: none;
    width: 8px;
    height: 8px;
    padding: 12px;
}
QScrollBar::right-arrow:horizontal {
    image: url(./res/rightarrow.png);
    border: none;
    width: 8px;
    height: 8px;
    padding: 12px;
}
QScrollBar::add-page:horizontal, QScrollBar::sub-page:horizontal {
    background: none;
}
```

Le definizioni per le barre di scorrimento orizzontali sono replicate da quelle verticali con le opportune modifiche del caso.

Definizione 7. QSizeGrip

```
QSizeGrip {
    image: url(./res/sizegrip.png);
    background: none;
}
```

Questa semplice definizione è necessaria per poter visualizzare, nell'angolo in basso a destra, le righe diagonali che segnalano la zona da trascinare per ridimensionare la finestra (la zona è visibile nella figura 5).

Definizione 8. QTabWidget

```
QTabWidget::pane {
    border: 1px solid #333333;
}
QTabWidget::tab-bar {
    left: 5px;
}
QTabWidget QToolButton {
    background-color: #444444;
    margin: 2px;
    padding: 1px;
}
QTabWidget QToolButton:hover {
    background-color: #555555;
}
```

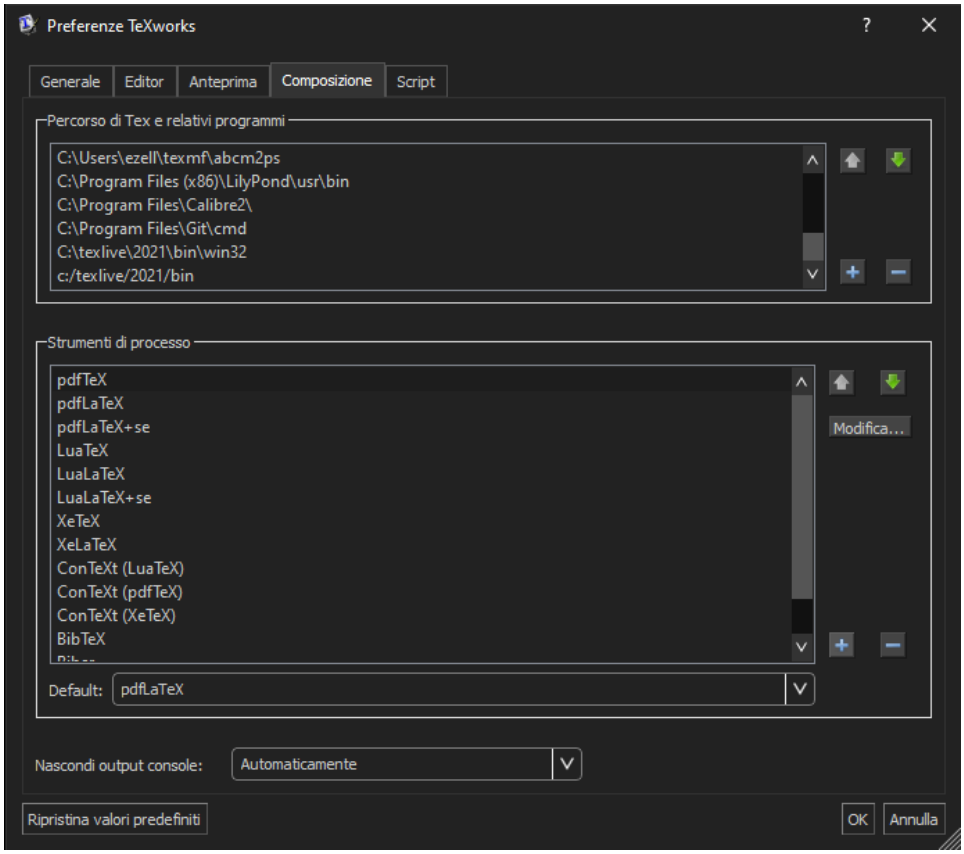

Figura 6. QTabWidget e QTabBar

`QTabWidget` è il riquadro che contiene le linguette che permettono di aprire schede diverse (come nella finestra delle preferenze, vedi la figura 6). È necessario inserire almeno la prima definizione per poter personalizzare anche `QTabBar`, l'elemento che controlla l'aspetto delle linguette vere e proprie. La seconda definizione sposta l'area delle linguette a sinistra di 5 pixel, soluzione che visivamente ho trovato preferibile.

Le ultime due definizioni impostano l'aspetto dei pulsanti all'interno di questa tipologia di riquadri; tra questi rientra anche il pulsante di chiusura della console, il quale, però, necessita di impostazioni diverse e verrà sistemato con la definizione 10.

Definizione 9. QTabBar

```
QTabBar::tab {
    background: #222222;
    border: 1px solid #555555;
    border-bottom-color: #333333;
    min-width: 8ex;
    padding: 4px 8px 4px 8px;
```

```
}
QTabBar::tab:selected {
    background: #333333;
    color: #ffffff;
    border-color: #555555;
    border-bottom-color: #333333;
}
QTabBar::tab:hover {
    background: #444444;
}
QTabBar::tab:!selected {
    margin-top: 2px;
}
```

Le prime due definizioni regolano l'aspetto delle linguette, rispettivamente nella versione normale e quando una linguetta è selezionata. Si noti come nella seconda ho modificato anche il colore del testo per farlo apparire più in risalto. Inoltre, in entrambe viene definito il colore del bordo, ma il colore del bordo inferiore è lo stesso impostato in `QTabWidget::pane`, ovvero il riquadro del contenuto della linguetta: in questo modo, la linguetta sembrerà un'estensione del riquadro (che è il comportamento che visivamente ci si aspetta). La terza definizione imposta anche un ulteriore colore di sfondo visibile al passaggio del mouse.

L'ultima definizione, assolutamente facoltativa, è un trucchetto carino: fa sì che tutte le linguette *non* selezionate siano più piccole di 2 pixel, mettendo così in risalto la linguetta selezionata.

Definizione 10. ClosableTabWidget

```
Tw--UI--ClosableTabWidget QToolButton {
    qproperty-icon: url(./res/close.png);
    background-color: #333333;
}
Tw--UI--ClosableTabWidget QToolButton:hover {
    background-color: #800000;
}
```

Queste due definizioni servono a modificare l'aspetto del pulsante di chiusura della console (figura 7). Appare subito evidente come il nome dell'elemento richiamato sia diverso da quelli visti finora. Questo perché tutti gli elementi modificati sono quelli standard previsti dal modello Qt, mentre `ClosableTabWidget` è un elemento creato dagli autori di TEXworks appositamente, quindi bisogna specificare il suo 'indirizzo' completo per poterlo richiamare (devo ringraziare uno degli autori, Stefan Löffler, che mi ha indicato il percorso corretto). Le definizioni vanno a modificare *tutti* i pulsanti di tipo `QToolButton` contenuti nell'elemento: al momento è uno solo, quello di chiusura, quindi alla versione 0.6.6 il codice è funzionante, ed è l'unico possibile. Dopo lo scambio con l'autore avvenuto su GitHub, posso anticipare che dalle prossime versioni il pulsante di chiusura avrà un nome proprio, un *id* (`closeButton`) che potrà essere richiamato e modificato direttamente, senza andare a toccare eventuali altri pulsanti che potrebbero venire aggiunti in futuro.

C'è un'altra differenza rispetto ai casi precedenti: per indicare un'immagine per il pulsante (anche in questo caso la croce originale è di colore scuro e non si vedrebbe con il nostro tema),

Figura 7. La console: il pulsante in rosso è ottenuto al passaggio del mouse

la proprietà `image` usata finora non funziona. Questo perché Qt non accetta da un foglio di stile esterno qualsiasi tipo di modifica a qualsiasi elemento, ma per ciascun elemento si possono modificare solo alcune proprietà prestabilite (si veda la documentazione riportata a inizio sezione). La soluzione è stata usare il comando `qproperty` che permette di modificare direttamente le proprietà di un elemento, tra cui l'immagine, e funziona anche dove non è previsto l'uso di `image`.

La seconda definizione imposta un colore di sfondo rosso scuro al passaggio del mouse.

Definizione 11. QDockWidget

```
QDockWidget {
    border: 1px solid lightgray;
    titlebar-close-icon: url(./res/close.png);
    titlebar-normal-icon: url(./res/undock.png);
}
QDockWidget::title {
    text-align: left;
    background: #222222;
    padding-left: 5px;
}
QDockWidget::close-button, QDockWidget::float-button {
    border: 1px solid transparent;
    background: none;
    padding: 0px;
}
QDockWidget::float-button:hover {
    background: #444444;
}
QDockWidget::close-button:hover {
    background: #800000;
}
```

`QDockWidget` è l'elemento che regola la finestra che contiene i risultati di una ricerca, o, per essere più specifici, la cornice della finestra con la barra del titolo (area in verde nella figura 8). La prima definizione imposta un bordo e le icone per i due pulsanti, quello di chiusura e quello per disancorare la finestra e renderla flottante (in azzurro nella figura 8). La seconda

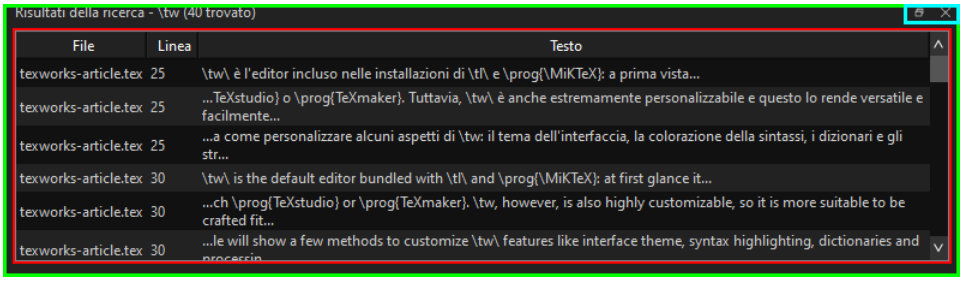

Figura 8. QDockWidget e la tabella definita da QHeaderView e QTableView

definizione regola l'aspetto della barra del titolo; la terza l'aspetto dei due pulsanti nel loro stato normale. La definizione raggruppa assieme due elementi: per Qt la sintassi è valida, l'importante è che a noi stia bene dare a due oggetti distinti le medesime caratteristiche. Trattandosi di due pulsanti, è corretto uniformarne l'aspetto; con le due seguenti definizioni invece ne differenzio il colore di sfondo al passaggio del mouse (preferisco sempre colorare di rosso i pulsanti di chiusura per evitare pressioni accidentali).

Definizione 12. QHeaderView & QTableView

```
QHeaderView::section  {
    background-color: #222222;
    border-right: 1px solid lightgray;
    border-left: 1px solid lightgray;
    color: white;
    padding-left: 4px;
}
QHeaderView::section:hover  {
    background-color: #444444;
}

QTableView {
   background-color: #111111;
   alternate-background-color: #222222;
   selection-background-color: #555555;
   font-color: #FFFFFF;
}
```

Ho raggruppato questi due elementi perché assieme definiscono l'aspetto dei risultati della ricerca, che sono inseriti in una tabella (l'area in rosso nella figura 8). `QHeaderView` stabilisce l'aspetto della riga di intestazione, mentre `QTableView` l'aspetto delle righe sottostanti. Qui è stata usata anche la proprietà `alternate-background-color` per indicare un colore diverso per le righe pari, così da creare un'alternanza tra i colori delle righe e migliorare la visibilità.

Definizione 13. QToolButton

```
QToolButton:hover {
    background-color: #444444;
}
```

Figura 9. QToolButton

Con questa definizione si imposta un colore che si attiva al passaggio del mouse per tutti i pulsanti con immagini, come quelli immediatamente sotto alle voci di menu principale (visibili nella figura 9).

Definizione 14. QPushButton

```
QPushButton {
    background-color: #222222;
    border-style: outset;
    border-width: 1px;
    border-color: #555555;
    padding: 4px;
}
QPushButton:pressed {
    background-color: #555555;
    border-style: inset;
    border-width: 1px;
    border-color: #555555;
    padding: 4px;
}
QPushButton:hover {
    background-color: #444444;
}
```

`QPushButton` invece definisce i pulsanti con contenuto testuale (figura 10). In questo contesto è stato definito lo stile del bordo, per rendere una sensazione di tridimensionalità dei pulsanti.

Definizione 15. ScreenCalibrationWidget

```
Tw--UI--ScreenCalibrationWidget {
    color: #000000;
}
```

Quest'ultima definizione si rivolge a un altro elemento specifico di TEXworks: il righello per la calibrazione dello schermo presente nelle preferenze (scheda 'anteprima'); come nel caso precedente, va indicato il percorso completo. Con la versione `0.6.6` di TEXworks il colore di sfondo del righello è bianco *hardcoded*, vale a dire che è definito direttamente nel codice del programma, e non si può modificare. Con questa definizione vado quindi a modificare in nero il colore di primo piano, che viene usato per disegnare le tacche e i numeri del righello. Ringraziando gli autori che hanno accolto il mio suggerimento, dalla prossima versione di TEXworks, anche il colore di sfondo del righello sarà modificabile e sarà quindi possibile renderlo coerente con il resto del tema.

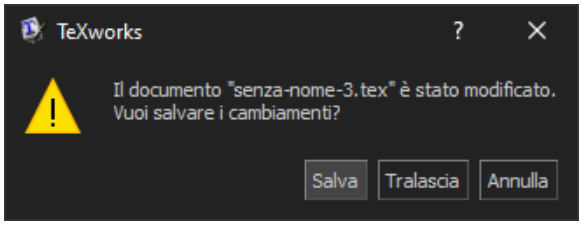

Figura 10. QPushButton

5.3. Suggerimenti

Tutte le definizioni viste finora permettono di ottenere un tema scuro funzionante; inoltre, le spiegazioni fornite a corredo dovrebbero essere sufficienti per consentire a chiunque di cimentarsi nel creare un tema colorato a piacimento. Prima di concludere, vorrei condividere alcuni suggerimenti frutto di molti esperimenti (ed errori!), che possono migliorare sensibilmente l'aspetto di un tema.

Il primo è la raccomandazione di usare il più possibile gli stati delle proprietà (*selected*, *pressed*, *hover*): in questo modo l'interfaccia apparirà dinamica, come se rispondesse alle attività dell'utente. Le prime versioni che ho creato non li prevedevano, e sembrava di avere tra le mani un programma vecchissimo e poco funzionante.

Il secondo consiglio riguarda la scelta della combinazione di colori. Chiaramente, andando a creare un tema ad uso personale, abbiamo la massima libertà di scelta. Tuttavia, credo sia comunque utile tenere in considerazione il rapporto di contrasto tra i colori desiderati, così da ottenere una maggiore leggibilità. Io mi sono rifatto alle raccomandazioni delle "Linee guida per l'accessibilità dei contenuti Web", le WCAG 2.1 (*Web Content Accessibility Guidelines*), dove, al criterio di successo 1.4.6 raccomandano un rapporto di contrasto di almeno 7 : 1. Esistono svariati strumenti online che permettono di calcolare il rapporto, io ho usato il seguente:

`https://juicystudio.com/services/luminositycontrastratio.php`

Un altro sito che uso spesso è `paletton.com`, uno strumento che permette, a partire dal codice `rgb` di un colore, di individuare, ad esempio, le gradazioni corrette da usare quando serve una variante più chiara o più scura (come per i cambi di stato di un elemento), oppure triadi (o anche tetradi) di colori abbinati tra loro secondo una serie di parametri configurabili (angolo di distanza nella ruota del colore, tonalità, saturazione, luminosità, contrasto).

6. La colorazione della sintassi

Una delle funzioni più utili di un editor di testo è la capacità di colorare la sintassi del linguaggio usato in un documento. In TEXworks abbiamo la possibilità di modificare le regole per la selezione della sintassi tramite delle *regular expression* (regex), di stabilire colori e formattazione personalizzati, e, volendo, anche di introdurre interi nuovi linguaggi. È utile approfondire la questione per almeno due motivi: da un lato la possibilità di modificare le regole ci permette di scegliere quale codice evidenziare, così da adattare il livello di dettaglio secondo le nostre preferenze; dall'altro la possibilità di modificare i colori è praticamente indispensabile se abbiamo

personalizzato l'interfaccia, perché tutte le regole predefinite usano tonalità pensate per un tema chiaro. In un'interfaccia con uno sfondo scuro i caratteri evidenziati risultano quasi invisibili.

Le regole che stabiliscono la colorazione sono salvate nel file `syntax-patterns.txt`, che si trova nella cartella `configuration`, la stessa che abbiamo già incontrato nella sezione 4. Il file che viene fornito assieme a TeXworks contiene già delle regole di base, suddivise in quattro linguaggi, chiamati: `LaTeX`, `LaTeX DTX`, `ConTeXt` e `BibTeX`. Qui prenderemo in esame solo il primo, ma una volta capito il meccanismo non sarà troppo difficile adattare le regole anche per altri tipi di file (cfr. nota 6). Il linguaggio predefinito `LaTeX` contiene cinque regole, che permettono di colorare: caratteri speciali, ambienti, pacchetti richiamati con `\usepackage`, comandi e commenti.

Nella prima parte del file è inoltre presente una sezione commentata che riporta già tutte le indicazioni di sintassi per inserire una regola. La struttura di base è la seguente:

⟨*stile testo*⟩ ⟨*controllo ortografico*⟩ ⟨*regex*⟩

Lo stile del testo prevede fino a tre tipi di definizioni:

1. Il colore del testo
2. Lo sfondo del testo
3. Un'indicazione della forma (*fontflag*):

   B grassetto (*bold*)
   I corsivo (*italic*)
   U sottolineato (*underlined*)

Il colore può essere indicato o con il nome secondo lo standard SVG, o con il valore `rgb`, nel formato esadecimale `#rrggbb` (in questo caso, va inserito uno spazio vuoto a inizio riga per evitare che la riga venga considerata un commento). La sintassi da usare è la seguente:

⟨*colore del testo*⟩/⟨*colore sfondo*⟩ ; ⟨*forma*⟩

La barra separa il colore del testo da quello di sfondo, il punto e virgola precede la forma del carattere.

Il controllo ortografico indica se per quella sezione di testo deve essere attivato il controllo ortografico, e va indicato solamente con `Y`(es) o `N`(o). In genere, sarà 'no' per tutte le definizioni, tranne che per quella dei commenti, solitamente scritti nella stessa lingua del documento.

L'ultima voce è la *regular expression*, la funzione che ha il compito di individuare una specifica stringa di testo nel documento: sarà proprio configurando tante *regex* che potremo assegnare colori e formattazioni ai vari elementi della sintassi LaTeX.

Prima di procedere con alcuni esempi, è bene ricordare che esiste una sorta di gerarchia tra le regole, stabilita dal loro ordine, per cui una regola più specifica va sempre inserita prima di una più generale.

Il primo esempio è la riga di definizione all'inizio della sezione del linguaggio `[LaTeX]`, la quale specifica di colorare di arancione i caratteri speciali.

```
# special characters
orange      N       [$#^_{}&]
```

In questa definizione sono chiaramente visibili le tre parti: colore, controllo ortografico e *regular expression*. Tra le parentesi quadre sono racchiusi tutti i caratteri da individuare.

Applicando la stessa logica, possiamo provare a individuare anche altri elementi. Consiglio di lasciare inalterati i linguaggi predefiniti, e crearne uno personalizzato dove potremo eseguire tutte le prove che vogliamo, senza correre il rischio di danneggiare qualcosa. Il modo più rapido è quello di creare una copia di tutto il linguaggio `[LaTeX]` con le sue definizioni e rinominare la copia come preferiamo, ad esempio `[LaTeX_test]`.

Qui di seguito elencherò alcune definizioni da aggiungere a questo linguaggio, assieme a un commento su come ho costruito la stringa *regex*.

```
# LaTeX refs
darkseagreen    N   \\(?:ref|eqref|pageref)\s*\{[^}]*\}

# LaTeX labels
lightseagreen   N   \\(?:label)\s*\{[^}]*\}
```

Queste due definizioni colorano i comandi per ottenere i riferimenti incrociati: ho usato due sfumature diverse dello stesso colore, il verde, per distinguere il comando `\label` che genera le etichette, dagli altri che le richiamano. La stringa di ricerca è stata ottenuta in questo modo:

```
\\
% trova il carattere \.
% analogamente a LaTeX, per indicare in maniera letterale i caratteri speciali che svolgono una funzione
    particolare in una regular expression bisogna farli precedere dal backslash.

(?:ref|eqref|pageref)
% gruppo che trova alternativamente uno dei tre comandi indicati. Il segno | funziona come l'operatore
    booleano OR. Questo è un gruppo 'non-capturing' ovvero che trova senza catturare, senza
    assegnare un ID.

\s*
% trova uno spazio (comprese tabulazioni e fine riga). Il carattere * è un quantificatore: recupera la
    ricerca che lo precede da 0 a infinite volte.

\{
% trova il carattere {.

[^}]*
% trova un qualsiasi carattere che non sia } da 0 a infinite volte.

\}
% trova il carattere }.
```

La stringa per le etichette è composta in maniera analoga.

Per chi si volesse cimentare nella creazione di stringhe personalizzate, consiglio il sito:

`https://regex101.com/`

La schermata principale presenta un riquadro in alto che permette di costruire una stringa di ricerca, un riquadro sottostante dove è possibile inserire del testo per verificare il funzionamento della stringa e una barra laterale a destra che contiene in alto uno strumento di analisi della stringa costruita e in basso un sintetico riepilogo dei comandi e dei *token* disponibili per costruire le stringhe. È uno strumento davvero utile che permette di risparmiare tantissimo tempo.

Vediamo ora la stringa per evidenziare i comandi di sezionamento.

```
# LaTeX sections
limegreen   N   \\(?:chapter|section|subsection|subsubsection)\s
    *(?:\[[^]]*\]\s*)?\{[^}]*\}
```

Come nell'esempio precedente, analizziamo la struttura della stringa.

```
\\
```
% trova il carattere \.

```
(?:chapter|section|subsection|subsubsection)
```
% gruppo che trova una tra le parole proposte.

```
\s*
```
% torna in quasi tutte le definizioni, poiché permette uno spazio, che comprende anche il fine riga.

```
(?:\[[^]]*\]\s*)?
```
% altro gruppo che serve a trovare un eventuale argomento opzionale del comando di sezionamento: prima trova il carattere [poi trova un numero indefinito di caratteri che non siano], poi il carattere], anche in questo caso viene ammesso lo spazio, e infine c'è il quantificatore ? che ritrova l'argomento che lo precede 0 o 1 volta.

```
\{[^}]*\}
```
% analogamente, questa sequenza trova l'argomento obbligatorio: carattere {, numero indefinito di caratteri che non siano } e infine } che chiude il gruppo.

Quella che segue è la soluzione che ho trovato per i comandi di citazione; dovrebbe supportare sia la maggior parte dei comandi di biblatex sia quelli dell'accoppiata BibTeX–natbib.

```
# LaTeX cite
lightcoral  N   \\(?i)(paren|text|foot|smart|super|auto)?cite([pt
    ]|author|year|date|url|title)?[*]?\s*(\[[^]]*\]){0,2}\{[^}]*\}
```

La stringa è piuttosto lunga, proviamo a vederla nel dettaglio.

```
\\
```
% backslash di inizio comando.

```
(?i)
```
% questo non è un gruppo, ma un flag: indica di recuperare tutta la stringa che segue indipendentemente che sia scritta in maiuscolo o minuscolo (*case (I)nsensitive*). Questo perché biblatex prevede per i suoi comandi una variante con la maiuscola.

```
(paren|text|foot|smart|super|auto)?
```
% biblatex usa il comando `\cite` e dei suoi composti, questo gruppo può trovare il prefisso del composto, il quantificatore ? trova 0 o 1 occorrenza.

```
cite([pt]|author|year|date|url|title)?
```

```
% qui viene chiesto direttamente di trovare la parola cite, con un'eventuale suffisso, inserito nel gruppo
    tra parentesi: p, t, author e year sono usati da natbib, questi ultimi e i rimanenti anche da
    biblatex.[8]

[*]?
% Viene ricercato letteralmente il carattere asterisco, serve perché sia natbib sia biblatex prevedono
    delle varianti asteriscate di alcuni comandi.

\s*
% solito comando per gli spazi.

(\[[^]]*\]){0,2}
% un comando di citazione di biblatex può avere 0,1 o 2 argomenti opzionali tra quadre. La stringa qui
    sopra è composta quindi con un gruppo che cattura tutto ciò che è contenuto tra due quadre, e da
    un quantificatore tra graffe che indica il numero di occorrenze: da zero a due.

\{[^}]*\}
% anche qui viene ricercato il contenuto tra due parentesi graffe, che costituiscono l'argomento
    obbligatorio con la chiave del comando di citazione.
```

Concludo questa sezione di definizioni con una soluzione per gli ambienti matematici in linea, ricavata da una risposta su Stack Overflow (2013); chi volesse approfondire l'argomento è invitato a leggere la risposta originale sul sito.

```
# math inline
hotpink N   (?<!\\)((?<!\$)\$(?!\$))(.*?)(?<!\\)(?<!\$)\1(?!\$)
```

Va sottolineato, tuttavia, che la stringa al momento non funziona, perché il motore regex di TEXworks per ora non supporta la funzione *lookbehind*. TEXworks è costruito con il toolkit Qt, che ha un suo motore interno per le espressioni regolari: quello attuale si chiama QRegularExpression, che ha sostituito il precedente QRegExp. Sebbene il nuovo costituisca un notevole miglioramento del precedente, non è ancora un motore "PCRE" (*Perl-compatible regular expression*), e alcune espressioni continuano a non essere disponibili in TEXworks. Se la situazione in futuro dovesse cambiare, la stringa appena vista potrebbe diventare molto utile.

Nel frattempo, per evidenziare la matematica in linea, ho optato per la soluzione molto più banale di evidenziare di un colore sgargiante il simbolo del dollaro, così da individuarlo immediatamente in mezzo al testo.[9]

```
# special characters
hotpink     N   \$
```

Come potete notare, tutte le definizioni viste finora non includono mai un cambio del colore di sfondo perché personalmente lo trovo una distrazione, tuttavia è stato indicato come fare, e ognuno si senta libero di sperimentare.

8. Per il comando `\cites` usato da biblatex ho creato un secondo comando a parte, poiché non sono riuscito a individuare un modo efficace di combinare i due. Ammetto la mia scarsa conoscenza di regex quindi non escludo sia possibile integrarli.
9. Se volete usufruire di questa soluzione, dovete ricordarvi di eliminare il segno del dollaro dalla prima definizione vista in questa sezione, quella che permette di colorare i caratteri speciali.

Riferimenti bibliografici

Delmotte, A., Löffler, S. *et al.* (2021). *A short manual for TEXworks*. URL https://github.com/TeXworks/manual/releases/download/2021-03-08/TeXworks-manual-en.pdf.

Gregorio, E. (2010). «Introduzione a TEXworks». URL http://profs.sci.univr.it/~gregorio/introtexworks.pdf.

Stack Exchange (2017). «How can i set a dark theme in texworks?» https://tex.stackexchange.com/questions/383271/how-can-i-set-a-dark-theme-in-texworks. [Online; ultimo accesso 22 agosto 2021].

Stack Overflow (2013). «Regex to match latex equations». https://stackoverflow.com/questions/14182879/regex-to-match-latex-equations/14537848#14537848. [Online; ultimo accesso 22 agosto 2021].

WCAG 2.1 (2018). «Web content accessibility guidelines (WCAG) 2.1 (traduzione italiana)». https://www.w3.org/Translations/WCAG21-it/. [Online; ultimo accesso 22 agosto 2021].

Filippo Vomiero
Università degli studi di Padova
filippo.vomiero@unipd.it

Functions in TEX and Elsewhere

Jean-Michel Hufflen

Abstract After a short survey about the notion of function in Mathematics and Computer Science, we look into the relationships between (LA)TEX's commands and features associated with functions: processing arguments, possible side effects, … Then we explore the relationships between the notion of types and the different kinds of objects handled by (LA)TEX's commands.

Sommario Dopo una breve panoramica sulla nozione di funzione in Matematica e in Scienza dell'Informazione, esaminiamo la relazione tra i comandi (LA)TEX e le proprietà associate alle funzioni: elaborazione degli argomenti, possibili effetti collaterali, ecc. Quindi esploriamo la relazione tra la nozione di tipo e le differenti specie di oggetti gestite dai comandi (LA)TEX.

1. Introduction

The starting idea for this present article originates from Gregorio (2020a), presented at last GUIT[1] meeting, an English version also existing as Gregorio (2020b). At the beginning of this article, Enrico Gregorio explains that within LATEX3's terminology, there is a clear separation between *variables* and *functions*: the former are used as *containers* whereas the latter perform some *action* when they are called. Let us recall that the first step towards LATEX3's implementation is the expl3 package, providing a new programming interface for LATEX. In particular, new syntactic conventions can be put into action between the two commands `\ExplSyntaxOn` and `\ExplSyntaxOff` (The LaTeX3 project, 2021). Let us go back to variables and functions within expl3, this separation only affects internal definitions, not user-level commands, as explained in Gregorio (2020b). In fact, this distinction between containers and action-performers already existed within TEX's commands, but was not obvious, since identical notations are used for both.

Hereafter we propose a short survey of this notion of function in Mathematics and Computer Science, then we explore the relationships between functions and definitions expressed using constructs of (LA)TEX's language, such as `\def` or `\newcommand`. Often our examples from more 'classic' programming originate from functional languages, where the notion of function is obviously central. Besides, many programming languages use *types* within function definitions, we also sketch this notion and its relation with TEX's commands. Reading this article only requires basic knowledge in Mathematics and Computer Science. (LA)TEX's commands mentioned throughout this article are described in Knuth (1986); Mittelbach *et al.* (2004).

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

2. What is a function?

2.1. In Mathematics

Let us assume that readers are familiar with the notion of *sets* in Mathematics[2]. A *binary relation* $\mathcal{R}$ between two sets S_0 and S_1 is a subset of the cartesian product $S_0 \times S_1$. If for all element x_0 of S_0:

(i) either there is no element of S_1 associated with x_0,

(ii) or only one element of S_1 is,

then $\mathcal{R}$ is a **function** from S_0 to S_1, S_0 (resp. S_1) being the *set of departure* (resp. *destination*) of $\mathcal{R}$. Formally, this definition can be expressed by:

$$\forall (x_0, x_1), (y_0, y_1) \in \mathcal{R}, x_0 = y_0 \Rightarrow x_1 = y_1$$

Let us notice that this definition is the most widespread, but is not universal. For example, Grätzer (1979) and Mac Lane (1971) consider only the (ii) condition, that is, *applications* according to the most widespread terminology in Mathematics. So we can notice that this notion of function is subject to variation.

A function of *several* arguments is a function whose domain is a cartesian product. A zero-argument function $f_0 : \longrightarrow S_1$ may be viewed as $f_0 : \{\varnothing\} \longrightarrow S_1$, as mentioned by Grätzer (1979)[3].

Let us recall that 'classic' Mathematics are based on sets in the sense that basic mathematical objects are introduced as sets. For example, an (x_0, x_1) couple is formally defined as the set $\{\{x_0\}, \{x_0, x_1\}\}$. With this definition, we can show that two couples are equal if and only if their elements are equal and in the same order[4]. As another example, a real number can be defined as the set of its minoring rational numbers. A different approach, more based on a notion of *computation process*, has been coined with the *λ-calculus* (Church, 1941). Without going thoroughly into the basic definitions of this formalism, let us mention that a function such as:

$$\begin{array}{rccc} f : & \mathbb{R} & \longrightarrow & \mathbb{R} \\ & x & \mapsto & x + 1 \end{array}$$

would be denoted by $f = \lambda x : \mathbb{R} \,.\, x + 1$ according to conventions closely related to *typed* λ-calculus, and $f = \lambda x \,.\, x + 1$ in 'simple' λ-calculus, without additional type information. This 'λ' notation, allowing a function to be wholly specified by an expression, has intensively been used by *functional programming languages*, such as Lisp dialects (McCarthy, 1960) or more modern languages such as Standard ML (Paulson, 1996) or Haskell (Peyton Jones, 2003). Such languages allow a function to be an argument or result of a subprogram. For example, we can easily program a sort procedure parameterised by the relation order used for elements. In addition, modern versions of general-purpose programming languages such

2. Readers interested in going thoroughly into this notion of *sets* can report to Halmos (1987), very didactic.
3. Let us remark that this definition has been *exactly* implemented in some programming languages—e.g., Haskell (Peyton Jones, 2003)—: such a f_0 function is of type $() \rightarrow S_1$, where '$()$' denotes the *unit* type, containing only the $()$ value.
4. The proof is quite tedious but not really difficult.

as C++ (Stroustrup, 1991), Python (Lutz, 1996), and Java (Java, 1997) have included such functional expressions[5].

2.2. In Computer Science

2.2.1. Generalities

Calling a *function*, within programming languages used in Computer Science, is related to a *process*, that is, a fragment of a computer program *being executed*. More precisely, a function is a subprogram that *produces* a *result*, whereas other kinds of subprograms—procedures—just perform some effects. Let us notice that the C programming language (Kernighan and Ritchie, 1988) has reversed this view since a C program is a sequence of functions returning results, particular cases are functions that returns a result being the `void` type: they just return to the caller function, as procedures do within earlier programming languages such as Algol (Naur, 1960) or Pascal (Wirth, 1971). Functions used within Computer Science may be very different from mathematical functions, since they can *modify* their environment. As a very simple example, the following C function counts the number of its calls—the `nb_of_times` variable being defined at the top level—:

```
int nb_of_times = 0 ;

int plusone_plus(int x) {
  nb_of_times++ ; return x + 1 ;
}
```

Let us mention that functional programming languages aim to put into action functions in the mathematical sense of this word, without *side effects*—unlike *imperative languages* such as Algol, Pascal or C—, but in practice, most of functional programming languages allow assignments and physical change of structures, that is, *side effects*.

The previous example has a great drawback: since the `nb_of_times` variable is global, *any* subprogram can modify it with its own way. A better version, related to the same functionality, is given in Scheme by the following function, returning a linear list with the successor of a number, followed by the number of calls of this function. Within this version, the `nb-of-times` variable is *local*, installed at the *definition* of this `plusone-plus-s` function, and incremented at each call, then saved. Only this `plusone-plus-s` function can access to the `nb-of-times` variable, so this variable is *protected* against other access.

```
(define plusone-plus-s
  (let ((nb-of-times 0))
    (lambda (x)
      (set! nb-of-times
            (+ nb-of-times 1))
      (list (+ x 1) nb-of-times))))
```

For example, the first two calls of this function could be:

```
(plusone-plus-s 2021) ⟹ (2022 1)
(plusone-plus-s 82)   ⟹ (83 2)
```

5. A didactic introduction to these *functional expressions* used in Java 8 can be found in Liguori and Liguori (2014).

This example allows us to introduce the notion of *closure*. The `plusone-plus-s` function deals with a *local environment*, retaining the local variable `nb-of-times`. This local variable must not be cleaned by a garbage collector between successive applications of the `plusone-plus-p` function, and this function should be able to retrieve this variable whenever it is called, that is possible by means of a mechanism so-called *lexical closure*. Some languages do not provide closures, but modern ones—e.g., Python or Java—do it. Sometimes a lexical closure allows a local variable to be read, but not written: this is the case in Java; Python's modern versions defaults to this behaviour, even if additional declarations allow such local variables to be updated.

2.2.2. Functions vs other objects

On another point, the Scheme functional programming language allows us to emphasise a great difference, between *functions* and *special forms*, already sketched in Hufflen (2020). Let us recall that Scheme systematically uses prefixed forms, that is:

```
(* 16 10 (+ 1918 103))      ⟹ 323360
(if (= 2020 2021) 'ok 'ko) ⟹ ko
```

In the first example, a function—the product, denoted by `*`—is applied to numbers, which may be directly provided or be the result of a computation, e.g., `(+ 1918 103)` ⟹ `2021`. In such a case, Scheme begins with the evaluation of all the arguments of a function before applying it, that is, the strategy is based on *calls by value*[6]. In the second example, a condition is evaluated, and depending on this result, the second or third argument of the `if` special form is evaluated and gives the result of this special form. Let us remark that `if` cannot be a function in Scheme; otherwise, it would evaluate all its arguments, which would be catastrophic in an example such as:

```
(if (zero? j) #f (/ i j))
```

where '`#f`' denotes the *false* logical value on Scheme. We can intuit that this expression implements a protection against a division by zero. But if all the arguments of `if` were evaluated, the `(/ i j)` expression would be evaluated even if `j` was equal to zero. So, it is impossible, for a programming language, to express any construct by means of a function, if calls by value are used for functions. More generally, this notion of special form applies to the constructs predefined in a programming language, such as conditional or iterative constructs, etc.[7] Some languages—including Scheme— allows the definition of *macros*, in which case the arguments of a macro are taken *verbatim*, without evaluation, as mentioned in **?**h2020:

```
(define-syntax twice-m
  (syntax-rules ()
    ((twice-m x)
```

6. Other strategies, possibly based on variables' *addresses*, were used within the first programming languages, or are still in use nowadays, especially within object-oriented languages. This point is outside this article's scope.
7. The difference between functions and other constructs exists in languages other than Scheme, but within 'traditional' languages such as C, the latter are expressed by syntactic markers—e.g., '`if (. . .) . . . ; else . . . ;`'—so the difference may appear more clearly. On the contrary, the same syntax is used for all commands provided by TEX, as in Lisp dialects (including Scheme).

```
    (list (quote x) (quote x)))))
(twice-m (+ 2020 1)) ⟹
       ((+ 2020 1) (+ 2020 1))
```

2.2.3. Call by value or by need

Several recent functional programming languages —including Haskell—no longer use a call-by-value strategy, but *lazy evaluation* (or *call by need*). An argument of a function is evaluated only if need be, and is evaluated once in such a case. The main interest of this *modus operandi* is that it avoids the evaluation of a program that loops if its result is not needed. Let us consider these two examples in C:

```
int endlessly() {
  back : goto back ; (* Loops endlessly! *)
  return 2020 ;      (* Never reached. *)
}
int thisyearf(int x) { return 2021 ; }
```

We can observe that the result returned by the `thisyearf` function is independent of its argument's value, so evaluating it is not needed *stricto sensu*. However, the evaluation of the expression:

```
thisyearf(endlessly())
```

loops: C's call-by-value strategy causes the subexpression `endlessly()` to be evaluated… without returning any value. Now let us look at an analagous definition using Haskell:

```
thisyearf x = 2021
```

The expression `thisyearf (1 / 0)` returns 2021 and the division by zero has not been performed since the argument has not been evaluated. We do not want the reading of this article to be slowed down, so some complements about the lazy evaluation have been put at Appendix A. Hereafter we just mention that:

- the lazy evaluation allows the definition of infinite objects: if only a subpart is of interest, the rest can remain untouched (cf. Appendix A);
- according to such a strategy, a conditional construct '`if ... then ... else ...`' can be viewed as a function: the first argument is evaluated, and according to the result, either the second or the third is evaluated.

2.3. Functions and (LA)TEX

In practice, we can consider that the (LA)TEX's commands introduced by constructs such as `\def` in *Plain TEX* or `\newcommand` in LATEX are functions returning strings when expanded fully. Other data types are used within TEX—e.g., counters, dimensions, etc.—but they are defined by other constructs. Some commands may return an empty string, in particular the commands that perform *side effects*, e.g.:

```
\def\skipfootnote{%
 \addtocounter{footnote}{1}%
}
```

which skips a footnote number. This behaviour includes commands that performs new definitions on the fly, e.g.:

```
\def\title#1{\gdef\@title{#1}}
```

After running this `title` command, the `\@title` command[8] will be a container for the document's title. (LA)TEX's commands without argument introduced by `\def` or `\newcommand`—e.g., `\skipfootnote`—are closer to zero-argument functions than constants ('single' elements of a set), since they are evaluated as many times as they are called. They are closer to *processes* than *associations* of variables with values.

There exists some significant similarity between TEX's language and functional programming languages: the latter allows functions to build new functions, the former allows commands to run commands dynamically built. If the generated command's name is the result of some computation, the construct '`\csname`…`\endcsname`' allows this name to be specified.

As mentioned in Hufflen (2020), TEX is a *dynamic* language. There is no way to use lexical closures. Let us consider this example, close to what we proposed in our previous article—let us recall that if C is a counter, '`\the`C' yields its value—:

```
\newcounter{lastyearc}
\setcounter{lastyearc}{2020}
\edef\firstsentence{%
 We ain't in \thelastyearc, are we?\par}
\def\secondsentence{%
 We are in \thelastyearc, ain't we?\par}
\addtocounter{lastyearc}{1}
```

The `\secondsentence` command yields the paragraph:

We are in 2021, ain't we?

whereas the `\firstsentence` command's body is expanded as far as possible at definition-time:

We ain't in 2020, are we?

So the `\edef` command allows a lexical scope to be simulated—as we show in Hufflen (2020)—and prevents against a command's redefinition. But the *updates* of a single command are ignored, too. In addition, if an (LA)TEX command uses a local environment, changes on it are not saved when this command exits. In TEX, persistent updatings can be applied only on global definitions, as we do in our first example `plusone_plus` in C (cf. § 2.2.1).

As mentioned in Hufflen (2020), TEX's commands are closer to macros than functions since the arguments are taken *verbatim* and processed when the command's body is expanded. Such a *modus operandi* allows a command to be deferred until it is applied. Let us consider the following definition:

8. Let us recall that this `\@title` command is both internal—its name contains the '@' character—and global—it has been introduced by the `\gdef` construct, which is a shorthand for '`\global\def`'.

```
\def\apply#1#2{#1{#2}}
```

If the first argument of this `\apply` command is also a command, it is applied to the second argument: '`\apply{\uppercase}{ok}`' returns 'OK'. Applying a command by giving its name as a string, is possible, too:

```
\def\applywrtname#1#2{%
 \csname#1\endcsname{#2}%
}
```

in which case an example could be:

```
\applywrtname{uppercase}{guit}
```

which produces 'GUIT'. As in a macro, TEX command's arguments are evaluated as many times as they occur. Some workarounds allow multiple evaluations to be avoided, and a kind of call by value can sometimes be simulated by using the `\expandafter` command (Knuth, 1986, p. 213), which expands the first token after the second. A 'classic' example is:

```
\uppercase\expandafter{\romannumeral 753}
```

where the `\romannumeral` command is expanded before the `\uppercase` command is applied, so this last command correctly processes Roman numerals and the result is 'DCCLIII', as expected. If the `\expandafter` is removed, then the `\uppercase` command processes the group between braces *verbatim*—which returns this group unchanged—and the `\romannumeral` command is applied, so the result is 'dccliii'.

The use of a *box* as a container (Knuth, 1986, pp. 120–122) can allow multiple evaluations of the same argument to be avoided, as shown in Hufflen (2020):

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

It is more related to a call by need than a call by value. A command's arguments are not evaluated and the *decision* of making a box is taken inside the command's body, in which case an argument may be evaluated only once. In other words, we are very close to a lazy evaluation.

2.4. In LATEX3

As abovementioned, the first step towards the implementation of LATEX3 has consisted in designing a new programming interface for LATEX by means of the expl3 package, documented in The LaTeX3 project (2021). Let us go back to our purpose, this package provides interesting and useful *abstraction barriers* between user-level commands and auxiliary definitions. However, since *all* the definitions are expanded into 'basic' TEX's language, this expl3 package cannot add features impossible to put into action in 'basic' TEX. So LATEX3 remains a dynamic language—as TEX—even if it possible to force expansions at definition-time. The scope of variables is clearly specified, but there is no local remanent variables which would allow closures.

Concerning the management of functions' arguments, the expl3 package proposes *type specifiers* that allow expansion to be controlled. As in TEX, this system is more related to macros than functions, since the expansion level can be specified: one-level or recursively[9]; in other words, the expansion can be *limited*. As an interesting innovation, specifying calls by value is easier, since these specifiers can apply to any argument. In §2.3, we showed that the `\expandafter` command can simulate it, but in practice this strategy is suitable for a command's first argument. For the others, we may have to put an impressive number of occurrences of this command. From our point of view, the finer control of expanding arguments is actually a great contribution of LATEX3.

3. Types

3.1. In Computer Science

Within this short survey, we do not go thoroughly into theoretical aspects about type theory[10]; according to a basic approach, we only consider a type as a characterisation of possible values for expressions used throughout programs. Regarding how types are handled, programming languages can be divided into three categories:

typed any value belongs to a type;

strongly typed in addition, variables are given a type;

untyped all the possible values belong to one type.

These definitions are the most commonly used, although some variations may be observed according to literature sources. In addition, we make precise that:

- in some languages, a *type inference* mechanism allows end users to be discharged from mentioning types; for example, Haskell acknowledges the definition of a function by making precise the arguments' types and result's; nevertheless, types are omnipresent within such languages: if the type inference fails, the corresponding definition or expression is rejected[11];
- some languages—including Scheme—manage types *dynamically*: functions allow users to be informed about any value's type;
- *types* and type *errors* should not be confused: for example, some people consider that Scheme is untyped because it is typed dynamically, but some type errors may occur—e.g., if an arithmetic operation is applied to non-numeric values—; as another example, PL/1 (IBM System 360, 1968) used type declarations, but a rich collection of conversion functions allowed a subprogram to be applied to any value (!), so type clashes were *impossible* within this language.

9. … which is close to calls by value.
10. Interested readers can consult Collins (2012), recent and very well documented about historical aspects of this part of Theoretical Computer Science.
11. For example, if the `thisyearf` function—cf. § 2.2.3—is processed by Standard ML, it acknowledges this function by giving its type: '`'a -> int`', where '`'a`' stands for 'any type, denoted by `a`'.

3.2. Types and (LA)TEX

Obviously, TEX's kernel is not a strongly typed language. Defining new types is not allowed, either. Is TEX an untyped language? We do not think so. In addition to characters and strings, it handles numbers, counters, dimensions, registers, and tables. A command crashes if it is applied to an object being a wrong type.

Is TEX a language dynamically typed? Sometimes, but not always. There is no test functions as in Scheme—`number?`, `string?`, etc.—but in some cases, a workaround may allow us to guess an object's type. Any TEXpert knows how to check if a `\c` command exists:

```
\expandafter\ifx\csname c\endcsname\relax%
  … % If the \c command is undefined, define it
    % as the \relax command and expand
    % this first part.
  \else… % Expand this second part if the \c
         % command is already defined.
\fi
```

by using the construct '`\csname`…`\endcsname`', mentioned at § 2.3. This *modus operandi* has been abbreviated in LATEX by:

```
\@ifundefined\c{. . . }{. . . }
```

and improved in e-TEX by the constructs:

```
\ifcsname c\endcsname… % If \c is defined.
  \else… % If not, it remains undefined.
\fi
```

or `\ifdefined\c{. . . }{. . . }`, equivalent to the previous expression.

Let us now go back to something close to type-checking and as an example, let us assume that we are wondering if a LATEX command can be applied to a counter defined by means of the `\newcounter` command. Let `ct` be the first argument: if it is actually such a counter, the `\thect` command allows its value to be displayed, so we can check if this command exists. Let us notice that such a test is a kind of *false-biased Monte Carlo algorithm*[12]: it is very quick; if it returns *false*, we are sure that the `ct` counter does not exist; if it returns *true*, the `\thect` command may exist without connection with a `ct` counter, even if such a case rarely occurs in practice. Besides, this *modus operandi* works with LATEX's counters, not with TEX's counters introduced by the `\newcount` command.

In fact, checking non-string objects in (LA)TEX could be easier if TEX's kernel included an error-handling mechanism, because all the non-string objects handled by TEX can be accessed by the `\the` command, which returns a string representation of such an object. But the 'trick' we have recalled applies only to a command name, we cannot check the validity of a complete expression.

12. In Computing, a *Monte Carlo algorithm* is an algorithm—often randomised, but always quick—whose output may be incorrect with a certain probability, quite small in practice (Metropolis, 1987).

3.3. In LaTeX3

Obviously the designers of LaTeX3 have wanted to put into action a *strongly typed* language. As explained in Gregorio (2020b), this decision does not apply to user-level commands, but to the implementation of such commands. On another point, it is preferable for LaTeX3's syntax to be close to LaTeX's. So the types of variables are not given as annotations—like in C—but belongs to the names of variables. For example, `\g_example_title_tl`, where '`tl`' stands for 'token list'. More generally, the format for a variable name is:

$$\backslash\langle scope\rangle\_\langle prefix\rangle\_\langle proper\ name\rangle\_\langle type\rangle$$

where:

- ⟨*scope*⟩ is 'c' for a constant, 'g' for a global variable, 'l' for a local one;
- ⟨*prefix*⟩ is a package name;
- ⟨*proper name*⟩ is the 'actual' name of the variable;
- ⟨*type*⟩ denotes its type.

Function names are similar:

$$\backslash\langle prefix\rangle\_\langle proper\ name\rangle\_\langle signature\rangle$$

where ⟨*signature*⟩ is a sequence of characters denoting the type of the function's successive arguments. For example, the arguments of the `\seq_seq_slit:Nnn` function are respectively of types N, n, and n; that is, a single token and two braced lists of tokens. The type of a function's result is not given within its signature.

Type specifications are different for variables and functions. A variable can retain any object of TeX, that is why types such as '`int`'—for integers—or '`box`' are allowed. The letters used within function signatures are related either to the look of the corresponding argument—e.g., 'c' is for a braced argument—or its evaluation. As mentioned in §2.4, the expansion of an argument can be controlled by using:

- 'e' for consuming an expansion's result;
- 'f' for a recursive expansion, ending as soon as an unexpandable token is found;
- 'o' for a one-level expansion;
- 'x' for a compile-time expansion, as performed by TeX's `\edef` construct.

The expl3 package provides a rich collection of predefined types. We personally regret that there is no way to define new type of containers in order to retain more structured information. But maybe it is planned for future versions… The information provided by signatures is very precise, provided that the expansion mechanism of TeX is mastered by programmers. Checking the type of a result is both easy and tedious: there is no functions checking types, we have to retain a result into a variable whose type is given.

4. Conclusion

The purpose of this article is twofold. The first point is that there are some variations about terms frequently used within programming. The list of the programming languages we have cited throughout this article is obviously non-limitative, it shows how different the implementations of comparable notions are. As a consequence, the writers of a documentation should be very careful and precise about terminology. The second point, started in Hufflen (2020), consists of studying the programming in TEX, since its *modus operandi* is quite apart from the other programming paradigms. We have showed that about the notions of functions and types. We can be told that this language is very specialised and has been recognised very suitable for typesetting texts (!), but the need for functionalities more related to 'classic' programming has led to the coexistence of TEX and another programming language, the best example being LuaTEX (Hagen, 2006). In addition, some features very specific to (LA)TEX may be difficult to understand, so it may be useful to compare them with analogous functionalities within other languages. That is true about TEX's kernel language, but will probably be true for LATEX3's future language. The mechanisms put into action with this project are interesting and promising, especially if we consider the clear separation between the interface of commands and their implementation. But undoubtedly the behaviour put into action within LATEX3 will be very different to the other programming paradigms, too.

A. Lazy evaluation

As mentioned in §2.2.3, if a lazy-evaluation strategy is used, an argument of a function is evaluated only if need be. For example, let us consider the following definition:

```
succ2nd x y = y + 1
```

Evaluating '`succ2nd (1 / 0) 2020`' yields 2021 and the first argument of the `succ2nd` function—the expression '`(1 / 0)`'—has been untouched, whereas '`succ2nd 2020 (1 / 0)`' fails because the second argument needs to be evaluated in order to perform the addition.

We also mentioned that this approach allows the specification of *infinite* objects. Hereafter, we show the easiest way to see that:

- a structure may be evaluated only partially;
- a part of such a structure, if it is evaluated, is evaluated once.

A very simple example of an infinite object in Haskell is:

```
naturalnumberlist =
  let from n = n : from (n + 1)
  in from 0
```

The internal function `from` returns a list of all the natural numbers from `n`—where $n \in \mathbb{N}$—that is, `n` followed[13] by the list of all the natural numbers from `n + 1`. Obviously, the lists returned by the internal function `from`—including the `naturalnumberlist` variable's value—are infinite objects. Let us use the GHCi[14] compiler of Haskell and its `:sprint` tool, typing the

13. In Haskell, the '`:`' infix operator separates the first element of a list and the following ones.
14. **G**lasgow **H**askell **C**ompiler **I**nteractive. A didactic introduction to it can be found in O'Sullivan *et al.* (2010).

command '`:sprint naturalnumberlist`' causes the following output to be displayed:

```
naturalnumberlist = _
```

where '_' means that the expression has not been evaluated yet. Now let us access the elements Nos. 0 and 2 of this list[15]:

```
naturalnumberlist !! 0 ⟹ 0
naturalnumberlist !! 2 ⟹ 2
```

After these two evaluations, let us type again the GHCi command—'`:sprint naturalnumberlist`'—and the result is:

```
naturalnumberlist = 0 : 1 : 2 : _
```

When we access to a particular element—or more generally a finite part—of this list, we need to compute all the elements before it, but the elements located after it can remain unevaluated: that is expressed by the '_' notation for the part of the list that is not evaluated yet. Some elements could be evaluated in the future if we move forward in this list. Of course, evaluating `naturalnumberlist` itself would cause an infinite loop to occur.

Acknowledgements

I am very grateful to the anonymous referees of this article's first version, since I was given constructive comments and an Italian translation of my abstract. I also thank Denis Bitouzé for his valuable comments.

References

Church, Alonzo (1941). *The Calculi of Lambda-Conversion*. Princeton University Press.

Collins, Jordan E. (2012). *A History of the Theory of Types: Developments After the Second Edition of 'Principia Mathematica'*. Lambert Academic Publishing.

Grätzer, George (1979). *Universal Algebra*. Springer-Verlag, 2nd edition.

Gregorio, Enrico (2020a). «Funzioni e expl3». *ArsTEXnica*, **30**, pp. 38–50. In Proc. GUIT 2020 meeting.

— (2020b). «Functions and expl3». *TUGBoat*, **41** (3), pp. 299–307.

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

Halmos, Paul Richard (1987). *Naive Set Theory*. Undergraduate Texts in Mathematics. Springer-Verlag.

Hufflen, Jean-Michel (2020). «Which success for TEX as an old program?» *ArsTEXnica*, **30**, pp. 24–30. In Proc. GUIT 2020 meeting.

15. In Haskell, the positions inside a list are numbered from zero, as in C.

IBM System 360 (1968). *PL/1 Reference Manual*.

Java (1997). *The Source for Java™ Technology*. Documentation available at: http://java.sun.com.

Kernighan, Brian W. and Dennis M. Ritchie (1988). *The C Programming Language*. Prentice Hall, 2nd edition.

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

The LaTeX3 project (2021). *The LATEX3 Interfaces*. http://ctan.math.illinois.edu/macros/latex/contrib/l3kernel/interface3.pdf.

Liguori, Robert and Patricia Liguori (2014). *Java 8 Pocket Guide. Instant Help for Java Programmers*. O'Reilly.

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

Mac Lane, Saunders (1971). *Categories for the Working Mathematician*. Numero 5 in Graduate Texts in Mathematics. Springer-Verlag.

McCarthy, John (1960). «Recursive functions of symbolic expressions and their computation by machine, part I». *Communications of the ACM*, **3** (4), pp. 184–195.

Metropolis, N. (1987). «The beginning of the Monte Carlo method». *Los Alamos Science*, **15**, p. 125–130.

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, 2nd edition.

Naur, Peter (1960). «Report on the algorithmic language Algol 60». *Communications of the ACM*, **3** (5), pp. 299–314.

O'Sullivan, Bryan, John Goerzen and Don Stewart (2010). *Real World Haskell*. O'Reilly.

Paulson, Lawrence C. (1996). *ML for the Working Programmer*. Cambridge University Press, 2ª edizione.

Peyton Jones, Simon (ed.) (2003). *Haskell 98 Language and Libraries. The Revised Report*. Cambridge University Press.

Stroustrup, Bjarne (1991). *The C++ Programming Language*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 2nd edition.

Wirth, Niklaus (1971). «The programming language Pascal». *Acta Informatica*, **1** (1), pp. 35–63.

Jean-Michel Hufflen
FEMTO-ST (UMR CNRS 6174) & University of Bourgogne Franche-Comté,
16, route de Gray,
25030 BESANÇON CEDEX
FRANCE
jmhuffle@femto-st.fr

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 2022. 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.


10032
9 771828 236001