

FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.



Budapest University of Technology and Economics

Faculty of Electrical Engineering and Informatics

Department of Automation and Applied Informatics

Modular Interpreter Written in JavaScript

THESIS

By

Heilig Benedek

Supervisor

Kundra László János

December 4, 2016

Contents

Kivonat	4
Abstract	5
Introduction	6
1 Lexical analysis	9
1.1 First point	9
2 A dolgozat formai kivitele	10
2.1 A dolgozat kimérete	10
2.2 A dolgozat nyelve	10
2.3 A dokumentum nyomdatechnikai kivitele	10
3 A L^AT_EX-sablon használata	11
3.1 Címkék és hivatkozások	11
3.2 Ábrák és táblázatok	12
3.3 Felsorolások és listák	14
3.4 Képletek	14
3.5 Irodalmi hivatkozások	16
3.6 A dolgozat szerkezete és a forrásfájlok	19
3.7 Alapadatok megadása	20
3.8 Új fejezet írása	20
Köszönetnyilvánítás	21

Bibliography	22
Függelék	23
F.1 A TeXnicCenter felülete	23
F.2 Válasz az „Élet, a világmindenség, meg minden” kérdésére	24

HALLGATÓI NYILATKOZAT

Alulírott *Heilig Benedek*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózataán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, December 4, 2016

Heilig Benedek
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon \LaTeX alapú, a *TeXLive* \TeX -implementációval és a PDF- \LaTeX fordítóval működőképes.

Abstract

This document is a \LaTeX -based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* \TeX implementation, and it requires the PDF- \LaTeX compiler.

Introduction

The main goal of the thesis is to create a modular interpreter in the native language of the modern Internet, JavaScript. The idea behind an interpreter such as this is that usually for someone new to programming, it is a hassle to download and install the various compilers and IDEs needed to even begin the actual programming phase, and this could be made easier with web technologies.

Nowadays, everyone has a laptop and a phone capable of browsing the web, and all these devices have a browser to do just that. Most dynamic websites use JavaScript as a language to manipulate the page (to make it dynamic), so all these browsers have a JavaScript engine built into them. I intend to use this capability to bring a more seamless first experience to those who want to learn programming and are at the beginning of their journey.

This is why I set out to create an interpreter that is:

- Written in JavaScript, thus available to be used on every kind of device
- Modular, thus making the creators of educational tools easier by abstracting the common things, and providing an easy way to allow execution of code inside the browser in any custom language that has a module created for it

To execute code written in a specific language, either a compiler (that translates directly into machine code), or an interpreter is used (which translate the code to another language higher than machine code). JavaScript is an interpreted language, partly that is why it can be found in so many places and thus the interpreter described in this thesis will be a cross-interpreter - that is, it will translate from an interpreted language to another interpreted language. But before any interpretation can begin, the code has to go through some transformations, mainly lexical analysis and parsing.

Lexical analysis is basically turning the source code into tokens using basic pattern matching rules, which will make the parsing stage easier. A token has a value and a class - the pattern matching pairs the values to their classes. A good and widely used tool for pattern matching is regular expressions, which is available in JavaScript as part of the language, and thus the proposed interpreter shall use. In case the writer of the module needs more complex rules, the lexical analyser should provide a way to use simple states and define transitions between these states.

In the next stage called parsing (or syntactical analysis), the tokens are turned into a parse tree using simple rules. A parse tree represents the hierarchy of the matched rules. One common way to define these rules is BNF (Backus-Naur form), which is a language to describe grammars. The given grammar describes every type of sentence in the language. In the definitions, we should be able to use the token classes from the previous stage (rules should be automatically added to the grammar). The parse tree then can be used to create an AST (Abstract Syntax Tree), which then can be used to interpret the code itself by evaluating the nodes, which basically represent nested parameters, some with fixed values, and some that need some computation to produce a value (and of course all these can have side effects).

To create a language module, the necessary steps are:

1. Create the rules for the lexical analyser
2. Provide the grammar in BNF
3. Parse the code, and iterate through the AST

The last step is the actual interpretation, which can be a simple evaluation of an expression, or a small virtual machine that keeps track of functions, variables, scopes and etc. This is the part that requires most attention when writing a language module.

Of course, something like this has been attempted many times, there are numerous solutions to creating parsers, although most of them are not written using JavaScript, they are more native tools to generate parsers. The list includes GNU Bison ¹, Jison ² (a clone of Bison in JavaScript), YACC ³ and ANTLR ⁴. But out of all of the available works, the most exciting is Syntax ⁵, which has the same goals (parser in JavaScript) and the same things in mind (educational usage) and my thesis idea, and which was published just a week after I've handed in my thesis proposal. The author of Syntax describes his motives in the article and the main differences from my thesis are the following:

- He only set out to create a modular (language agnostic) parser
- He provides multiple parsing algorithms
- He provides multiple target languages, including JavaScript, but that only means that the parser that is generated will be a JavaScript module (other targets include Python, Ruby and PHP)
- He provides a way to set operator precedence

¹GNU Bison <https://www.gnu.org/software/bison/>

²Jison - Your friendly JavaScript parser generator! <http://zaa.ch/jison/>

³Yet Another Compiler-Compiler <http://dinosaur.compilertools.net/>

⁴Another Tool For Language Recognition <http://www.antlr.org/>

⁵Syntax - language agnostic parser generator <https://medium.com/@DmitrySoshnikov/syntax-language-agnostic-parser-generator-bd24468d7cfc>

- He does not provide a state based lexical analyser

In every other sense, the two projects are very similar, although I have only learned about Syntax during later stages of development.

Another similar project is my own project - from which the idea came for this thesis -, which is a Logo interpreter that translates to JavaScript, and which was based on absolutely no research about parsers, and thus is buggy but generally works quite well. It had two iterations, the first of which I built 4 years ago as a hobby project. The second iteration is also a hobby project, but it is much more mature, and can handle Logo source pretty well. The choice of language - Logo - is because it is a good step towards understanding of basic ideas of programming (in both functional and imperative paradigms), and also provides the user with something more materialistic (it is easy to create beautiful and complex drawings), which is great for young children to get a reward for programming, which is a great way to motivate them. The most beautiful things they can draw are fractals (up to a certain level of depth), which also teach them a lot about recursion, which can be hard to understand for new pupils (fractals are a good way to visualize recursion).

Other solutions to running code on the web include cloud based solutions, which means that the code that the user inputs is actually compiled and ran on a remote virtual machine. While this is great in the sense that it is exactly as the "real thing", it has a great drawback - it requires an active internet connection to use. My solution can be bundled into a desktop application (or the website can be downloaded) using tools like the electron ⁶ framework. This allows the students to practice while on the go (and without an active internet connection), which cloud based solutions can not allow.

During the implementation, I will not be focusing on optimal speeds, since this is not a tool built for performance but rather one that is built to be available everywhere. To achieve modern, well readable source code and portability, I will be using the ECMAScript 6 coding standard, which is available in some modern browsers, but not so many older ones. I will develop my solution in a Node.js module, which can then be converted into code that works on older browsers using Babel ⁷ and Browserify ⁸, which are Node.js modules to convert to older standards and to bundle modules into one source file for browsers respectively.

In Chapter 1, I will describe the implemented lexical analyser, and it's inner workings. In Chapter 2, I will write about the parser module, and the implemented parsing algorithm. In Chapter 3, I will describe the two example modules I have created - a math expression parser, and a simple Logo subset interpreter.

⁶Electron <http://electron.atom.io/>

⁷Babel <https://github.com/babel/babel>

⁸Browserify <http://browserify.org/>

Chapter 1

Lexical analysis

1.1 First point

asdada

Chapter 2

A dolgozat formai kivitele

Az itt található információk egy része a BME VIK Hallgatói Képviselőlet által készített „Utolsó félév a villanykaron” c. munkából lett kis változtatásokkal átemelve. Az eredeti dokumentum az alábbi linken érhető el: <http://vik-hk.bme.hu/diplomafelev-howto-2009>.

2.1 A dolgozat kimérete

A minimális 50, az optimális kiméret 60-70 oldal (függelékkel együtt). A bírálók és a záróvizsga bizottság sem szereti kifejezetten a túl hosszú dolgozatokat, így a bruttó 90 oldalt már nem érdemes túlszárnyalni. Egyébként függetlenül a dolgozat kiméretétől, ha a dolgozat nem érdekfeszítő, akkor az olvasó már az elején a végét fogja várni. Érdemes zárt, önmagában is érthető művet alkotni.

2.2 A dolgozat nyelve

Mivel Magyarországon a hivatalos nyelv a magyar, ezért alapértelmezésben magyarul kell megírni a dolgozatot. Aki külföldi posztgraduális képzésben akar részt venni, nemzetközi szintű tudományos kutatást szeretne végezni, vagy multinacionális cégnél akar elhelyezkedni, annak célszerű angolul megírnia diplomadolgozatát. Mielőtt a hallgató az angol nyelvű verzió mellett dönt, erősen ajánlott mérlegelni, hogy ez mennyi többletmunkát fog a hallgatónak jelenteni fogalmazás és nyelvhelyesség terén, valamint - nem utolsó sorban - hogy ez mennyi többletmunkát fog jelenteni a konzulens illetve bíráló számára. Egy nehezen olvasható, netalán érthetetlen szöveg teher minden játékos számára.

2.3 A dokumentum nyomdatechnikai kivitele

A dolgozatot A4-es fehér lapra nyomtatva, 2,5 centiméteres margóval (+1 cm kötésbeni), 11-12 pontos betűmérettel, talpas betűtípussal és másfeles sorközzel célszerű elkészíteni.

Chapter 3

A L^AT_EX-sablon használata

Ebben a fejezetben röviden, implicit módon bemutatjuk a sablon használatának módját, ami azt jelenti, hogy sablon használata ennek a dokumentumnak a forráskódját tanulmányozva válik teljesen világossá. Amennyiben a szoftver-keretrendszer telepítve van, a sablon alkalmazása és a dolgozat szerkesztése L^AT_EX-ben a sablon segítségével tapasztalataink szerint jóval hatékonyabb, mint egy WYSWYG (*What You See is What You Get*) típusú szövegszerkesztő esetén (pl. Microsoft Word, OpenOffice).

3.1 Címkék és hivatkozások

A L^AT_EX dokumentumban címkéket (`\label`) rendelhetünk ábrákhoz, táblázatokhoz, fejezetekhez, listákhoz, képletekhez stb. Ezekre a dokumentum bármely részében hivatkozhatunk, a hivatkozások automatikusan feloldásra kerülnek.

A sablonban makrókat definiáltunk a hivatkozások megkönnyítéséhez. Ennek megfelelően minden ábra (*figure*) címkéje **fig**: kulcsszóval kezdődik, míg minden táblázat (*table*), képlet (*equation*), fejezet (*section*) és lista (*listing*) rendre a **tab**:, **eq**:, **sect**: és **listing**: kulcsszóval kezdődik, és a kulcsszavak után tetszőlegesen választott címke használható. Ha ezt a konvenciót betartjuk, akkor az előbbi objektumok számára rendre a `\figref`, `\tabref`, `\eqref`, `\sectref` és `\listref` makrókkal hivatkozhatunk. A makrók paramétere a címke, amelyre hivatkozunk (a kulcsszó nélkül). Az összes említett hivatkozástípus, beleértve az `\url` kulcsszóval bevezetett web-hivatkozásokat is a **hyperref**¹ csomagnak köszönhetően aktívak a legtöbb PDF-nézegetőben, rájuk kattintva a dokumentum megfelelő oldalára ugrik a PDF-néző vagy a megfelelő linket megnyitja az alapértelmezett böngészővel. A **hyperref** csomag a kimeneti PDF-dokumentumba könyvjelzőket is készít a tartalomjegyzékből. Ez egy szintén aktív tartalomjegyzék, amelynek elemeire kattintva a nézegető behozza a kiválasztott fejezetet.

¹Segítségével a dokumentumban megjelenő hivatkozások nem csak dinamikussá válnak, de színezhetőek is, bővebbet erről a csomag dokumentációjában találunk. Ez egyúttal egy példa lábjegyzet írására.

3.2 Ábrák és táblázatok

A képeket PDFLaTeX esetén a veszteségmentes PNG, valamint a veszteséges JPEG formátumban érdemes elmenteni. Az EPS (PostScript) vektorgrafikus képformátum beillesztését a PDFLaTeX közvetlenül nem támogatja. Ehelyett egy lehetőség 200 dpi, vagy annál nagyobb felbontásban raszterizálni a képet, és PNG formátumban elmenteni. Az egyes képek mérete általában nem, de sok kép esetén a dokumentum összmérete így már szignifikáns is lehet. A dokumentumban felhasznált képfájlokat a dokumentum forrása mellett érdemes tartani, archiválni, mivel ezek hiányában a dokumentum nem fordul újra. Ha lehet, a vektorgrafikus képeket vektorgrafikus formátumban is érdemes elmenteni az újrafelhasználhatóság (az átszerkeszthetőség) érdekében.

Kapcsolási rajzok legtöbbször kimásolhatók egy vektorgrafikus programba (pl. CorelDraw) és onnan nagyobb felbontással raszterizálva kimenthetők PNG formátumban. Ugyanakkor kiváló ábrák készíthetők Microsoft Visio vagy hasonló program használatával is: Visio-ból az ábrák közvetlenül PNG-be is menthetők.

Lehetőségeink Matlab ábrák esetén:

- Képernyőlopás (*screenshot*) is elfogadható minőségű lehet a dokumentumban, de általában jobb felbontást is el lehet érni más módszerrel.
- A Matlab ábrát a **File/Save As** opcióval lementhetjük PNG formátumban (ugyanaz itt is érvényes, mint korábban, ezért nem javasoljuk).
- A Matlab ábrát az **Edit/Copy figure** opcióval kimásolhatjuk egy vektorgrafikus programba is és onnan nagyobb felbontással raszterizálva kimenthetjük PNG formátumban (nem javasolt).
- Javasolt megoldás: az ábrát a **File/Save As** opcióval EPS *vektorgrafikus* formátumban elmentjük, PDF-be konvertálva beillesztjük a dolgozatba.

Az EPS kép az `epstopdf` programmal² konvertálható PDF formátumba. Célszerű egy batch-fájlt készíteni az összes EPS ábra lefordítására az alábbi módon (ez Windows alatt működik).

```
@echo off
for %%j in (*.eps) do (
echo converting file "%%j"
epstopdf "%%j"
)
echo done .
```

Egy ilyen parancsfájlt (`convert.cmd`) elhelyeztük a sablon `figures\eps` könyvtárába, így a felhasználónak csak annyi a dolga, hogy a `figures\eps` könyvtárba kimenti az EPS

²a korábban említett L^AT_EX-disztribúciókban megtalálható

formátumú vektorgrafikus képet, majd lefuttatja a `convert.cmd` parancsfájlt, ami PDF-be konvertálja az EPS fájlt.

Ezek után a PDF-ábrát ugyanúgy lehet a dokumentumba beilleszteni, mint a PNG-t vagy a JPEG-et. A megoldás előnye, hogy a lefordított dokumentumban is vektorgrafikusan tárolódik az ábra, így a mérete jóval kisebb, mintha raszterizáltuk volna beillesztés előtt. Ez a módszer minden – az EPS formátumot ismerő – vektorgrafikus program (pl. CorelDraw) esetén is használható.

A képek beillesztésére az ?? fejezetben mutattunk be példát (?? ábra). Az előző mondatban egyúttal az automatikusan feloldódó ábrahivatkozásra is láthatunk példát. Több képfájl is beilleszthetünk egyetlen ábrába. Az egyes képek közötti horizontális és vertikális margót metrikusan szabályozhatjuk (3.1. ábra). Az ábrák elhelyezését számtalan tipográfiai szabály egyidejű teljesítésével a fordító maga végzi, a dokumentum írója csak preferenciáit jelezheti a fordító felé (olykor ez bosszúságot is okozhat, ilyenkor pl. a kép méretével lehet játszani).

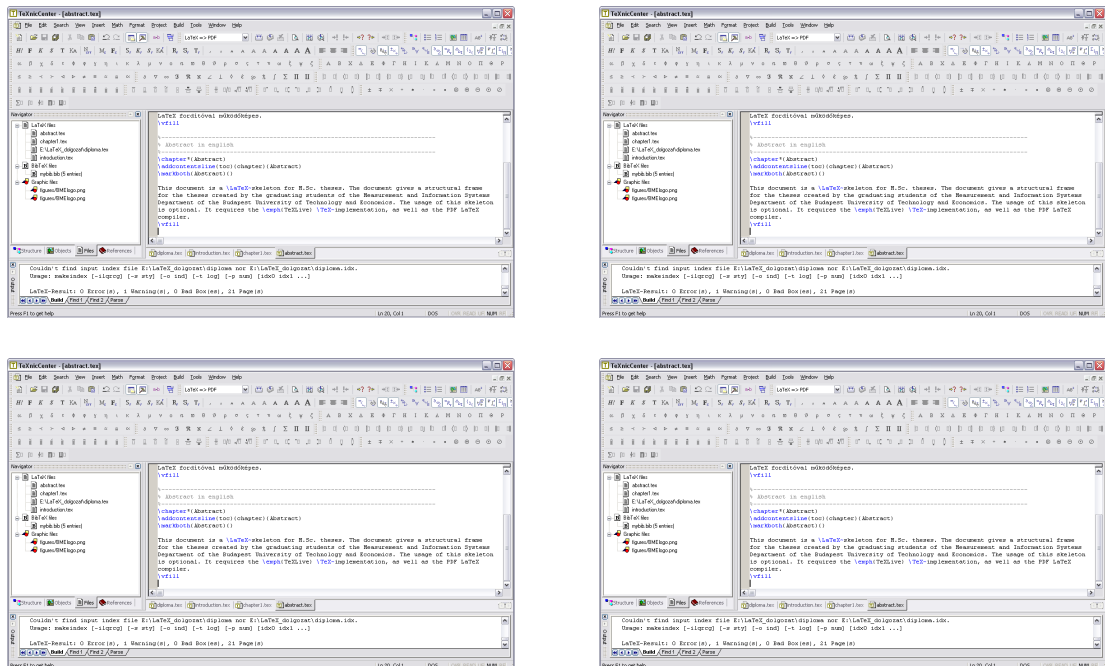


Figure 3.1: Több képfájl beillesztése esetén térközoeket is érdekes használni.

A táblázatok használatára a 3.1. táblázat mutat példát. A táblázat címkéje nem véletlenül került a táblázat fölé, ez a szokványos.

Table 3.1: Az órajel-generátor chip órajel-kimenetei.

Órajel	Frekvencia	Cél pin
CLKA	100 MHz	FPGA CLK0
CLKB	48 MHz	FPGA CLK1
CLKC	20 MHz	Processzor
CLKD	25 MHz	Ethernet chip
CLKE	72 MHz	FPGA CLK2
XBUF	20 MHz	FPGA CLK3

3.3 Felsorolások és listák

Számozatlan felsorolásra mutat példát a jelenlegi bekezdés:

- *első bajusz*: ide lehetne írni az első elem kifejtését,
- *második bajusz*: ide lehetne írni a második elem kifejtését,
- *ez meg egy szakáll*: ide lehetne írni a harmadik elem kifejtését.

Számozott felsorolást is készíthetünk az alábbi módon:

1. *első bajusz*: ide lehetne írni az első elem kifejtését, és ez a kifejtés így néz ki, ha több sorosra sikeredik,
2. *második bajusz*: ide lehetne írni a második elem kifejtését,
3. *ez meg egy szakáll*: ide lehetne írni a harmadik elem kifejtését.

A felsorolásokban sorok végén vessző, az utolsó sor végén pedig pont a szokásos írásjel. Ez alól kivételt képezhet, ha az egyes elemek több teljes mondatot tartalmaznak.

Listákban a dolgozat szövegétől elkülönítendő kódrészleteket, programsorokat, pseudo-kódokat jeleníthetünk meg (3.1. lista). A lista keretét, háttérszínét, egész stílusát megválaszthatjuk.

lista 3.1: A fenti számozott felsorolás \LaTeX - forráskódja

```
\begin{enumerate}
  \item \emph{első bajusz:} ide lehetne írni az első elem kifejtését,
    és ez a kifejtés így néz ki, ha több sorosra sikeredik,
  \item \emph{második bajusz:} ide lehetne írni a második elem kifejtését,
  \item \emph{ez meg egy szakáll:} ide lehetne írni a harmadik elem kifejtését.
\end{enumerate}
```

Ráadásul különféle programnyelveket és a nyelveken belül kulcsszavakat is definiálhatunk, ha szükséges. Erről bővebbet a `listings` csomag hivatalos leírásában találhatunk.

3.4 Képletek

Ha egy formula nem túlságosan hosszú, és nem akarjuk hivatkozni a szövegből, mint például a $e^{i\pi} + 1 = 0$ képlet, *szövegekőzi képletként* szokás leírni. Csak, hogy másik példát is lássunk, az $U_i = -d\Phi/dt$ Faraday-törvény a $\text{rot } E = -\frac{dB}{dt}$ differenciális alakban adott Maxwell-egyenlet felületre vett integráljából vezethető le. Látható, hogy a \LaTeX -fordító a sorközoeket betartja, így a szöveg szedése esztétikus marad szövegekőzi képletek használata esetén is.

Képletek esetén az általános konvenció, hogy a kisbetűk skalárt, a kis félkövér betűk (\mathbf{v}) oszlopvektort – és ennek megfelelően \mathbf{v}^T sorvektort – a kapitális félkövér betűk (\mathbf{V}) mátrixot jelölnek. Ha ettől el szeretnénk térni, akkor az alkalmazni kívánt jelölésmódot célszerű

külön alfejezetben definiálni. Ennek megfelelően, amennyiben \mathbf{y} jelöli a mérések vektorát, ϑ a paraméterek vektorát és $\hat{\mathbf{y}} = \mathbf{X}\vartheta$ a paraméterekben lineáris modellt, akkor a *Least-Squares* értelemben optimális paraméterbecslő $\hat{\vartheta}_{LS} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ lesz.

Emellett kiemelt, sorszámozott képleteket is megadhatunk, ennél az `equation` és a `eqnarray` környezetek helyett a korszerűbb `align` környezet alkalmazását javasoljuk (több okból, különféle problémák elkerülése végett, amelyekre most nem térünk ki). Tehát

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \quad (3.1)$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}, \quad (3.2)$$

ahol \mathbf{x} az állapotvektor, \mathbf{y} a mérések vektora és \mathbf{A} , \mathbf{B} és \mathbf{C} a rendszert leíró paraméter-mátrixok. Figyeljük meg, hogy a két egyenletben az egyenlőségjelek egymáshoz igazítva jelennek meg, mivel a mindkettőt az `&` karakter előzi meg a kódban. Lehetőség van számozatlan kiemelt képlet használatára is, például

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}.$$

Mátrixok felírására az $\mathbf{A}\mathbf{x} = \mathbf{b}$ inhomogén lineáris egyenlet részletes kifejtésével mutatunk példát:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \quad (3.3)$$

A `\frac` utasítás hatékonyságát egy általános másodfokú tag átviteli függvényén keresztül mutatjuk be, azaz

$$W(s) = \frac{A}{1 + 2T\xi s + s^2 T^2}. \quad (3.4)$$

A matematikai mód minden szimbólumának és képességének a bemutatására természetesen itt nincs lehetőség, de gyors referenciaként hatékonyan használhatók a következő linkek:

http://www.artofproblemsolving.com/LaTeX/AoPS_L_GuideSym.php,

<http://www.ctan.org/tex-archive/info/symbols/comprehensive/symbols-a4.pdf>,

<ftp://ftp.ams.org/pub/tex/doc/amsmath/short-math-guide.pdf>.

Ez pedig itt egy magyarázat, hogy miért érdemes `align` környezetet használni:

<http://texblog.net/latex-archive/maths/eqnarray-align-environment/>.

3.5 Irodalmi hivatkozások

Egy \LaTeX dokumentumban az irodalmi hivatkozások definíciójának két módja van. Az egyik a `\thebibliography` környezet használata a dokumentum végén, az `\end{document}` lezárás előtt.

```
\begin{thebibliography}{9}

\bibitem{Lamport94} Leslie Lamport, \emph{\LaTeX: A Document Preparation System}.
Addison Wesley, Massachusetts, 2nd Edition, 1994.

\end{thebibliography}
```

Ezek után a dokumentumban a `\cite{Lamport94}` utasítással hivatkozhatunk a forrásra. A fenti megadás viszonylag kötetlen, a szerző maga formázza az irodalomjegyzéket.

Egy sokkal professzionálisabb módszer a BiBTeX használata, ezért ez a sablon is ezt támogatja. Ebben az esetben egy külön szöveges adatbázisban definiáljuk a forrásmunkákat, és egy külön stílusfájl határozza meg az irodalomjegyzék kinézetét. Ez, összhangban azzal, hogy külön formátumkonvenció határozza meg a folyóirat-, a könyv-, a konferenciakik-stb. hivatkozások kinézetét az irodalomjegyzékben (a sablon használata esetén ezzel nem is kell foglalkoznia a hallgatónak, de az eredményt célszerű ellenőrizni). A felhasznált hivatkozások adatbázisa egy `.bib` kiterjesztésű szöveges fájl, amelynek szerkezetét a 3.2. kódrészlet demonstrálja. A forrásmunkák bevitelkor a sor végi vesszők külön figyelmet igényelnek, mert hiányuk a BiBTeX -fordító hibaüzenetét eredményezi. A forrásmunkákat típus szerinti kulcsszó vezeti be (`@book` könyv, `@inproceedings` konferenciakiadványban megjelent cikk, `@article` folyóiratban megjelent cikk, `@techreport` valamelyik egyetem gondozásában megjelent műszaki tanulmány, `@manual` műszaki dokumentáció esetén stb.). Nemcsak a megjelenés stílusa, de a kötelezően megadandó mezők is típusról-típusra változnak. Egy jól használható referencia a <http://en.wikipedia.org/wiki/BibTeX> oldalon található.

A stílusfájl egy `.sty` kiterjesztésű fájl, de ezzel lényegében nem kell foglalkozni, mert vannak beépített stílusok, amelyek jól használhatók. Ez a sablon a BiBTeX -et használja, a hozzá tartozó adatbázisfájl a `mybib.bib` fájl. Megfigyelhető, hogy az irodalomjegyzéket a dokumentum végére (a `\end{document}` utasítás elé) beillesztett `\bibliography{mybib}` utasítással hozhatjuk létre, a stílusát pedig ugyanitt a `\bibliographystyle{plain}` utasítással adhatjuk meg. Ebben az esetben a `plain` előre definiált stílust használjuk (a sablonban is ezt állítottuk be). A `plain` stíluson kívül természetesen számtalan más előre definiált stílus is létezik. Mivel a `.bib` adatbázisban ezeket megadtuk, a BiBTeX -fordító is meg tudja különböztetni a szerzőt a címtől és a kiadótól, és ez alapján automatikusan generálódik az irodalomjegyzék a stílusfájl által meghatározott stílusban.

Az egyes forrásmunkákra a szövegből továbbra is a `\cite` paranccsal tudunk hivatkozni, így a 3.2. kódrészlet esetén a hivatkozások rendre `\cite{Wettl04}`, `\cite{Candy86}`,

lista 3.2: *Példa szöveges irodalomjegyzék-adatbázisra BiBTeX használata esetén.*

```
@BOOK{Wettl04,
  author="Ferenc Wettl and Gyula Mayer and Péter Szabó",
  title="\LaTeX`kézikönyv",
  publisher="Panem Könyvkiadó",
  year=2004
}
@ARTICLE{Candy86,
  author = "James C. Candy",
  title = "Decimation for Sigma Delta Modulation",
  journal="{IEEE} Trans.\ on Communications",
  volume = 34,
  number = 1,
  pages = "72--76",
  month = jan,
  year = 1986,
}
@INPROCEEDINGS{Lee87,
  author = "Wai L. Lee and Charles G. Sodini",
  title = "A Topology for Higher Order Interpolative Coders",
  booktitle = "Proc.\ of the IEEE International Symposium on
Circuits and Systems",
  year = 1987,
  vol = 2,
  month = may # "~4--7",
  address = "Philadelphia, PA, USA",
  pages = "459--462"
}
@PHDTHESIS{KissPhD,
  author = "Peter Kiss",
  title = "Adaptive Digital Compensation of Analog Circuit Imperfections
for Cascaded Delta-Sigma Analog-to-Digital Converters",
  school = "Technical University of Timi\c{s}oara, Romania",
  month = apr,
  year = 2000
}
@MANUAL{Schreier00,
  author = "Richard Schreier",
  title = "The Delta-Sigma Toolbox v5.2",
  organization = "Oregon State University",
  year = 2000,
  month = jan,
  note = "\newline URL: http://www.mathworks.com/matlabcentral/fileexchange/"
}
@MISC{DipPortal,
  author="Budapesti {M}űszaki és {G}azdaságtudományi {E}gyetem
{V}illamosmérnöki és {I}nformatikai {K}ar",
  title="{D}iplomaterv portál (2011 február 26.)",
  howpublished="\url{http://diplomaterv.vik.bme.hu/}",
}}

```

`\cite{Lee87}`, `\cite{KissPhD}`, `\cite{Schreier00}` és `\cite{DipPortal}`. Az irodalomjegyzékben alapértelmezésben csak azok a forrásmunkák jelennek meg, amelyekre található hivatkozás a szövegben, és ez így alapvetően helyes is, hiszen olyan forrásmunkákat nem illik az irodalomjegyzékbe írni, amelyekre nincs hivatkozás.

Mivel a fordítási folyamat során több lépésben oldódnak fel a szimbólumok, ezért gyakran többször (TeXLive és TeXnicCenter esetén 2-3-szor) is le kell fordítani a dokumentumot. Ilyenkor ez első 1-2 fordítás esetleg szimbólum-feloldásra vonatkozó figyelmeztető üzenettel zárul. Ha hibaüzenettel zárul bármelyik fordítás, akkor nincs értelme megismételni, hanem a hibát kell megkeresni. A `.bib` fájl megváltoztatáskor sokszor nincs hatása a változtatásnak azonnal, mivel nem mindig fut újra a BibTeX fordító. Ezért célszerű a változtatás után

azt manuálisan is lefuttatni (TeXnicCenter esetén **Build/BibTeX**).

Hogy a szövegbe ágyazott hivatkozások kinézetét demonstráljuk, itt most sorban meghivatkozunk a [10], [3], [6], [5] és az [8] forrásmunkát, valamint az [7] weboldalt.

Megjegyzendő, hogy az ékezetes magyar betűket is tartalmazó **.bib** fájl az **inputenc** csomaggal betöltött **latin2** betűkészlet miatt fordítható. Ugyanez a **.bib** fájl hibaüzenettel fordul egy olyan dokumentumban, ami nem tartalmazza a `\usepackage[latin2]{inputenc}` sort. Speciális igény esetén az irodalmi adatbázis általánosabb érvényűvé tehető, ha az ékezetes betűket speciális latex karakterekkel helyettesítjük a **.bib** fájlban, pl. á helyett `\'a`-t vagy ő helyett `\H{o}`-t írunk.

Oldaltörés következik (ld. forrás).

3.6 A dolgozat szerkezete és a forrásfájlok

A diplomatervsablon (a kari irányelvek szerint) az alábbi fő fejezetekből áll:

1. 1 oldalas *tájékoztató* a szakdolgozat/diplomaterv szerkezetéről (`guideline.tex`), ami a végső dolgozathoz törlendő,
2. *feladatkiírás* (`project.tex`), a dolgozat nyomtatott verziójában ennek a helyére kerül a tanszék által kiadott, a tanszékvezető által aláírt feladatkiírás, a dolgozat elektronikus verziójában pedig a feladatkiírás egyáltalán ne kerüljön bele, azt külön tölti fel a tanszék a diplomaterv-honlapra,
3. *címoldal* (`titlepage.tex`),
4. *tartalomjegyzék* (`diploma.tex`),
5. a diplomatervező *nyilatkozata* az önálló munkáról (`declaration.tex`),
6. 1-2 oldalas tartalmi *összefoglaló* magyarul és angolul, illetve elkészíthető még további nyelveken is (`abstract.tex`),
7. *bevezetés*: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása (`introduction.tex`),
8. sorszámmal ellátott *fejezetek*: a feladatkiírás pontosítása és részletes elemzése, előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések, a tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása, a megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek (`chapter{1,2..n}.tex`),
9. esetleges *köszönetnyilvánítások* (`acknowledgement.tex`),
10. részletes és pontos *irodalomjegyzék* (ez a sablon esetében automatikusan generálódik a `diploma.tex` fájlban elhelyezett `\bibliography` utasítás hatására, a 3.5. fejezetben leírtak szerint),
11. *függelékek* (`appendices.tex`).

A sablonban a fejezetek a `diploma.tex` fájlba vannak beillesztve `\include` utasítások segítségével. Lehetőség van arra, hogy csak az éppen szerkesztés alatt álló `.tex` fájlt fordítsuk le, ezzel lerövidítve a fordítási folyamatot. Ezt a lehetőséget az alábbi kódrészlet biztosítja a `diploma.tex` fájlban.

Ha az alábbi kódrészletben az egyes sorokat a `%` szimbólummal kikommentezzük, akkor a megfelelő `.tex` fájl nem fordul le. Az oldalszámok és a tartalomjegyek természetesen csak akkor billennek helyre, ha a teljes dokumentumot lefordítjuk.

```

\includeonly{
    guideline,%
    project,%
    titlepage,%
    declaration,%
    abstract,%
    introduction,%
    chapter1,%
    chapter2,%
    chapter3,%
    acknowledgement,%
    appendices,%
}

```

3.7 Alapadatok megadása

A diplomaterv alapadatait (cím, szerző, konzulens, konzulens titulusa) a `diploma.tex` fájlban lehet megadni az alábbi kódrészlet módosításával.

```

\newcommand{\vikszerzo}{Bódis-Szomorú András}
\newcommand{\vikkonzulens}{dr.~Konzulens Elemér}
\newcommand{\vikcim}{Elektronikus terelők}
\newcommand{\viktanszek}{Méréstechnika és Információs Rendszerek Tanszék}
\newcommand{\vikdoktipus}{Diplomaterv}
\newcommand{\vikdepartmentr}{Bódis-Szomorú András}

```

3.8 Új fejezet írása

A főfejezetek külön `chapter{1..n}.tex` fájlban foglalnak helyet. A sablonhoz 3 fejezet készült. További főfejezeteket úgy hozhatunk létre, ha új `chapter{i}.tex` fájlt készítünk a fejezet számára, és a `diploma.tex` fájlban, a `\include` és `\includeonly` utasítások argumentumába felvesszük az új `.tex` fájl nevét.

Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Bibliography

- [1] Another tool for language recognition. <http://wwwantlr.org/>.
- [2] Gnu bison. <https://www.gnu.org/software/bison/>.
- [3] James C. Candy. Decimation for sigma delta modulation. *IEEE Trans. on Communications*, 34(1):72–76, January 1986.
- [4] Jison - your friendly javascript parser generator! <http://zaa.ch/jison/>.
- [5] Peter Kiss. *Adaptive Digital Compensation of Analog Circuit Imperfections for Cascaded Delta-Sigma Analog-to-Digital Converters*. PhD thesis, Technical University of Timișoara, Romania, April 2000.
- [6] Wai L. Lee and Charles G. Sodini. A topology for higher order interpolative coders. In *Proc. of the IEEE International Symposium on Circuits and Systems*, pages 459–462, Philadelphia, PA, USA, May 4–7 1987.
- [7] Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Kar. Diplomaterv portál (2011 február 26.). <http://diplomaterv.vik.bme.hu/>.
- [8] Richard Schreier. *The Delta-Sigma Toolbox v5.2*. Oregon State University, January 2000. URL: <http://www.mathworks.com/matlabcentral/fileexchange/>.
- [9] Dmitry Soshnikov. Syntax - language agnostic parser generator. <https://medium.com/@DmitrySoshnikov/syntax-language-agnostic-parser-generator-bd24468d7cfc>.
- [10] Ferenc Wettl, Gyula Mayer, and Péter Szabó. *LaTeX kézikönyv*. Panem Könyvkiadó, 2004.
- [11] Yet another compiler-compiler. <http://dinosaur.compilertools.net/>.

Függelék

F.1 A TeXnicCenter felülete

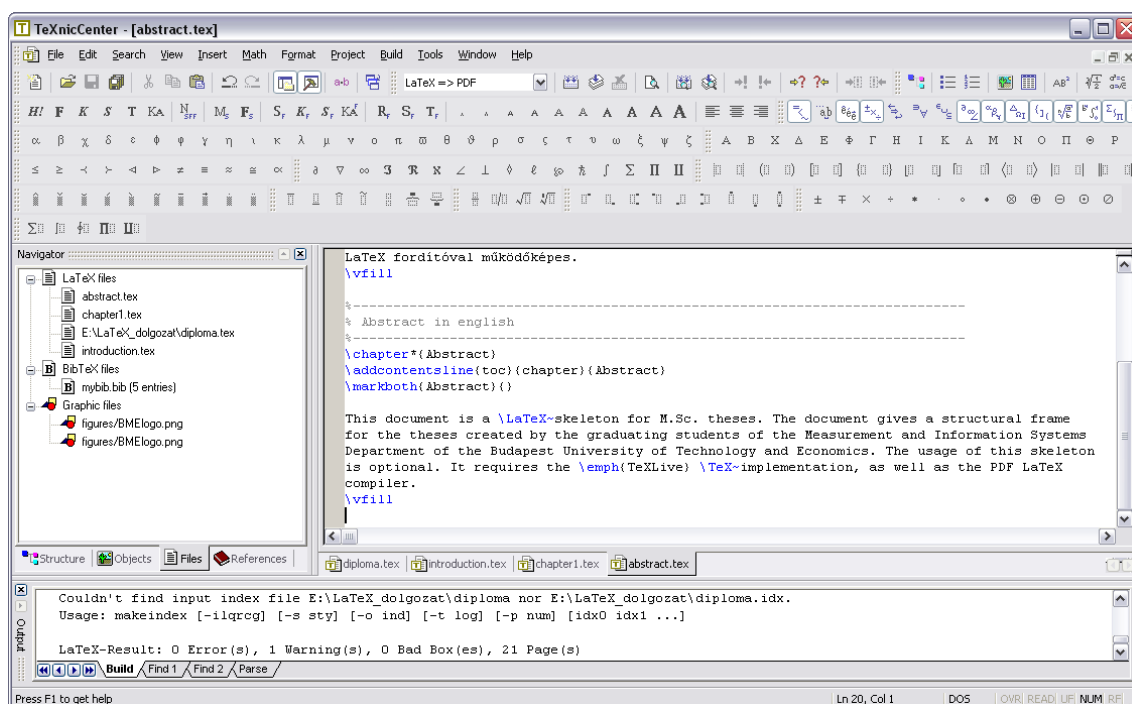


Figure F.1.1: A TeXnicCenter Windows alapú \LaTeX -szerkesztő.

F.2 Válasz az „Élet, a világmindenség, meg minden” kérdésre

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42. \quad (\text{F.2.1})$$

A Faraday-indukciós törvényből levezetve

$$\text{rot } E = -\frac{dB}{dt} \quad \longrightarrow \quad U_i = \oint_{\mathbf{L}} \mathbf{E} d\mathbf{l} = -\frac{d}{dt} \int_A \mathbf{B} d\mathbf{a} = 42. \quad (\text{F.2.2})$$