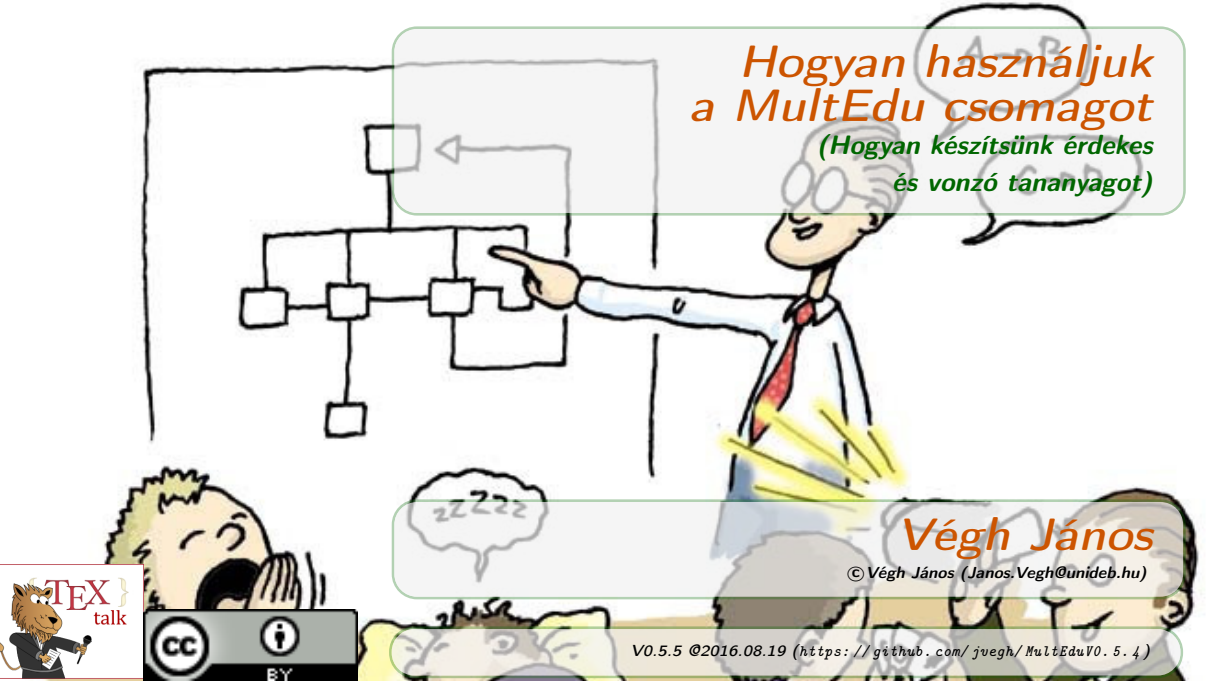
Hogyan használjuk a MultEdu csomagot

(Hogyan készítsünk érdekes és vonzó tananyagot)



Végh János

© Végh János (Janos.Vegh@unideb.hu)

V0.5.5 ©2016.08.19 (https://github.com/jvegh/MultEduV0.5.4)

Manapság a tananyagot a hallgatóság változatos formákban igényli: előadáson nagy méretű, jól áttekinthető, kivetíthető anyagot kell használni, amely képekkel gazdagon illusztrált és az előadó számára is jó sorvezetőként szolgál. A vizsgára készüléshez pedig arra a magyarázatra is szükség van, amit az előadó a kivetített anyaghoz élő szóban hozzáfűz.
A jelen dokumentum egyúttal bemutató és tulajdonság tesztelő is. A dokumentum megkísérli bemutatni, mit hogyan kell és lehet használni, egyúttal azt is megvizsgálva, hogy tényleg működik-e az elvárt módon. A sokféle tulajdonság és különösen azok kölcsönhatása miatt sok munkát és időt igényel a fejlesztés, ezért a tényleges tulajdonságok nem mindig egyeznek meg a dokumentációval, különösen a kezdeti fázisban.

A makró csomag (legalább) három különböző felhasználói szinten alkalmazható. Már a legalacsonyabb szinten is szükségesek a LaTeX-re vonatkozó elemi ismeretek. Az alap szinten a felhasználó egyszerűen csak helyettesíti és módosítja a rendszert bemutató dokumentumokat. Haladó szinten (ehhez már el kell olvasni a felhasználói leírást is ☺) megtanulja a csomagban található makrók által biztosított lehetőségeket, és azokat aktívan használva fejleszti dokumentumait. Tapasztalt felhasználóként saját makrókat is készíthet (jó, ha azokat a letölthető anyaghoz hozzáadja), azaz aktívan részt vesz a fejlesztésben.

A MultEdu (mint minden LaTeX alapú rendszer) feltételezi, hogy a felhasználó már rendelkezik tapasztalatokkal a LaTeX használatában. Azaz, a felhasználó rendszerén már működnie kell valamilyen LaTeX rendszernek. Az egyszerű használat és a gyors elindulás érdekében célszerű a lentebb megadott módon saját projekt csoportjainak egy főkönyvtárat és azon belül az egyes projekteknek alkönyvtárakat létrehozni. A leggyorsabb magát a `./Workstuff` könyvtárat (a megfelelő átnevezésekkel és törlésekkel) lemásolni, és csekély módosításokkal elkészíteni saját 'Helló Világ' programját. Ezután érdemes legalább átlapozni a felhasználói kézikönyvet, ami után már elkezdheti saját fejlesztését. Eleinte csak szöveget, aztán sorjában megtanulni a használni kívánt tulajdonságok programozását.

A MultEdu rendszert az alábbi könyvtár szerkezetben célszerű használni. Két fő könyvtára: a `./common`, amely tartalmazza a MultEdu összes fájlját, és a `./Workstuff`, amely a felhasználói könyvtár szerkezetet modellezi.

```
.
|-- common
|-- WorkStuff
```

A felhasználói projekt csoportokat ilyen szerkezetben érdemes hozzáadni:

```
.
|-- Exams
|-- Labs
|-- Lectures
|-- Papers
```

amely könyvtáraknak a `|-- WorkStuff` könyvtárhoz hasonló belső alkönyvtárai vannak

A ./common különleges célú al-alkönyvtárakat, valamint általános célú makró fájlokat tartalmaz.

```
.
|-- common
| |-- defaults
| |-- formats
| |-- images
```

A ./defaults al-alkönyvtár olyan alapértelmezett szöveget tárol, mint a szerzői jogok.

A ./formats al-alkönyvtár tartalmazza a formátumokat meghatározó makrókat

Az ./images al-alkönyvtár képeket tartalmaz

A ./Workstuff al-alkönyvtár tartalmazza (a példa programként is szolgáló) felhasználói leírás fájljait. Egy olyan ./Workstuff/Demo projektet tartalmaz, amelyik (a saját főkönyvtárában) három fájlból áll.

```
|-- WorkStuff
|  |-- Demo
|  .  |-- CMakeLists.txt
|  .  |-- Demo.tex
|  .  |-- Main.tex
```

A valódi főprogram Main.tex, és ehhez készült egy Demo.tex nagyon egyszerű boríték. Ha használja a UseLATEX csomagot, a Main.tex file használata (ezzel a névvel) kötelező, a boríték fájl nevét pedig a CMakeLists.txt fájllal egyeztetni kell.

Célszerű a felhasználói projekt könyvtárakat is hasonlóan berendezni.

```
|-- WorkStuff
| |-- Demo
| . |-- build
| . . . |-- build
| . |-- dat
| . |-- fig
| . |-- lst
| . |-- src
```

A fő `Main.tex` menet közben magába olvassa az alkönyvtárakban levő egyéb fájlokat.

| . |-- `src` tartalmazza a felhasználó forráskód fájljait,
| . |-- `fig` a képeit,
| . |-- `lst` a programlisták forrás kódját,
| . |-- `dat` a többi adatot .

A CMake rendszeren keresztül a UseLATEX csomag is használható arra, hogy egy szerkesztés után, a kötegelt feldolgozási módot használva, egyetlen lépésben elő lehessen állítani a forrásnyelvi fájlból a különböző nyelvű és formátumú dokumentumokat; erre való a `CMakeLists.txt` fájl.
A

```
|-- build
```

és

```
| . . |-- build
```

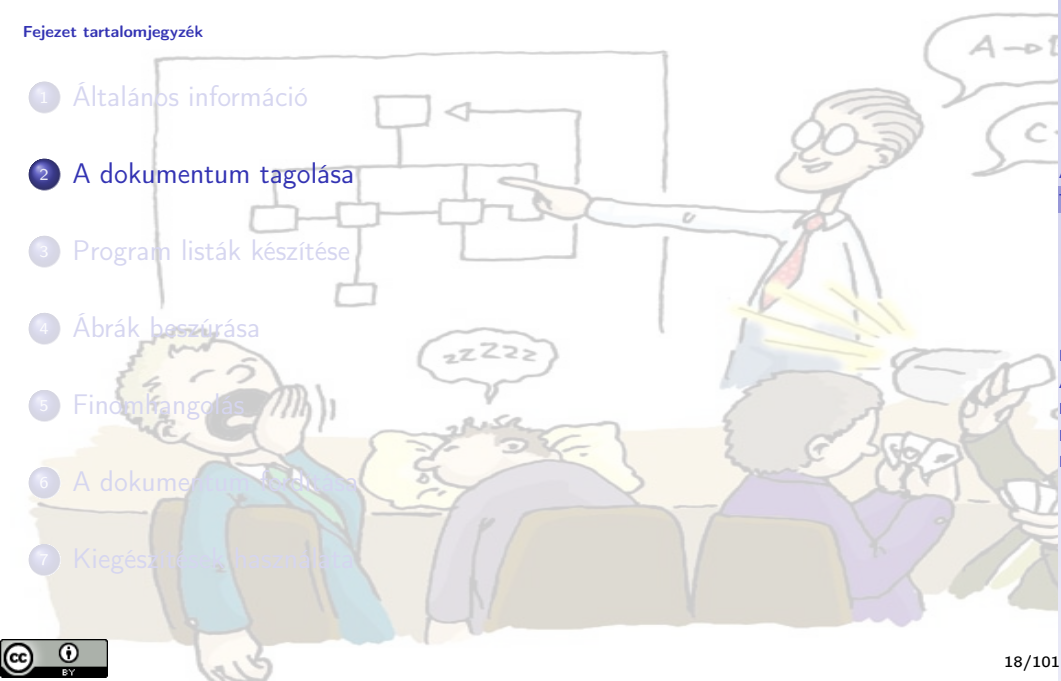
alkönyvtárak csak akkor kellenek ha a CMake rendszert használjuk.

A fordítás során a LaTeX számos munka fájlt állít elő. Ezek sajnos a projekt gyökér könyvtárába kerülnek. Amint a 2 szakaszban látható, a működéshez csak 3 fájl szükséges, a többi bármikor törölhető.
A kötegelt feldolgozás is készít a projekt gyökér könyvtárába `.tex` forrás fájlokat. Ezek is bármikor törölhetők, de akár 'kézi' fordítással kimenő fájlt is készíthetünk belőlük. Ez utóbbi esetben érdemes előtte az `src/Defines.tex` fájlt átszerkeszteni.

A MultEdu csomag teljes forráskódot tartalmaz (szépítgetés nélkül). A szerző nem LaTeX szakértő, csak régi felhasználó. A makrók nagy része adaptált az Interneten megtalálható eredeti forrásokból. A forráskód tartalmazza a hivatkozást az eredeti kódra, a felhasználói kézikönyv nem veszteget helyet köszönetnyilvánításra. A szerző azonban köszönetét fejezi ki az eredeti szerzőknek, mint az eredeti kódért, mind a különböző felhasználói közösségekben nyújtott támogatásárt.

A csomag tartalmaz pár .pdf fájlt, különböző formátumban és nyelven. A fájl nevében nem szerepel a verzió szám (a címlapon igen). Eme fájlok célja (amellett, hogy felhasználói kézikönyvként is szolgálnak), hogy a leendő felhasználók gyorsan fel tudják mérni, ilyen tulajdonságokkal rendelkező dokumentáló rendszert akarnak-e.

A MultEdu makró csomagot úgy tettem közzé, ahogy van ('as is'). Folyamatosan és egyenetlenül fejlesztem, én magam már jól tudok vele tananyagot fejleszteni. A makrókat és a dokumentációt is fejlesztem, de az (sok) időt igényel. Működési és dokumentációs hibák leírását, még esetleges tulajdonságok fejlesztésének kérését is örömmel fogadom.



A dokumentumot a 'beamer' csomag követelményeinek megfelelően kell szervezni. A nyomtatható formában való megjelenítéshez a MultEdu a 'beamerarticle' csomagot használja, és a tagolást is megfelelően változtatni kell. Ennek érdekében a MultEdu saját tagolási egységeket vezet be, amelyek valójában a 'book' formátumnak felelnek meg és amelyeket dia készítéshez megfelelően átalakít. A szöveg viszont 'diakeret' egységekből áll össze.

Használata:
`\MEframe[keys]{subtitle}{content}`
Értelmezett kulcsok
`shrink=true|false` and `plain=true|false`
Alapértelmezetten mindkettő false.

A dokumentum legnagyobb egysége a fejezet.
Használata:
`\MEchapter[short title]{long title}`
Ha diákat készítünk, `\section` lesz belőle.

A következő, kisebb egység a szakasz. Használata:
`\MEsection[r´Ovid cím]{hosszú cím}`
Hasonló módon létezik
`\MEsubsection [r´Ovid cím] {hosszú cím}`
és
`\MEsubsubsection [r´Ovid cím] {hosszú cím}`;
ez utóbbi dia készítés esetén `\paragraph` alakot ölt.

Előfordul, hogy ugyanazt az anyagot saját nyelvemen oktatom hazai hallgatóknak, és angolul, külföldi hallgatóknak. A tananyag megegyezik, és együtt kell fejleszteni. Nyilván előnyös, ha a két anyag ugyanabban a forrásnyelvi fájlban, egymás mellett fejleszthető.
Erre szolgál a `\UseSecondLanguage`. A fent bevezetett makróknak van egy 'D' (Dual) taggal kibővített változata, amelyikben mind az elsődleges, mind a másodlagos nyelven megadjuk a szükséges tartalmakat.

Használata:
\UseSecondLanguage{YES}
ahol az {}-ben megjelenő argumentum nem számít, csak az, hogy definiálva van-e ez a makro.
A kétféle makrókészlet keverhető, de csak a 'D' makrók reagálnak a nyelv változtatásra.

Kétnyelvű dokumentumokban általában a
`\MEDframe[keys]{subtitle, first language} {content, first language } {subtitle, second language} {content, second language}`
keretet használjuk. Azaz a felhasználó megadja mindkét nyelven a címet és a tartalmat, majd fordítás előtt `\UseSecondLanguage` használatával kiválasztja az egyik nyelvet.

Bár tananyag készítésekor kevésbé fontos szempont, egy konferencia előadás bemutatásakor nagyon fontos az előadásra szánt idő megfelelő felhasználása. A MultEdu a kivetített diákon a felhasznált idő kivetítésével tudja ezt támogatni. Ez a lehetőség alapállapotban tiltott, külön engedélyezni kell `\def\EnableTimer{YES}` utasítással, célszerűen a `src/Defines.tex` fájlban. Ezt az utasítást célszerű az első "valódi" keret címében elhelyezni, különben üres keretet eredményezhet. Példa:
`\MEframe{Keret cim \ifx\EnableTimer\undefined \else \initclock\fi}`
A MultEdu a kijelzett idő színének megváltoztásával figyelmezteti az előadót, ha az előadás végéhez közel kerül. A maximális értéket
`\def\LectureTime{perc}`
utasítással lehet beállítani, az alapértelmezett érték 15.

Hasonlóképpen, a kétnyelvű dokumentum legnagyobb egysége a 'Dchapter'. (Amint említettük, dia készítéskor ez átalakul 'Dsection' egységgé.)
Használata:
`\MEDchapter[r˝Ovid cím1]{hosszú cím1}{r˝Ovid cím2}{hosszú cím2}`
ami aztán átalakul
`\MEchapter[r˝Ovid cím1]{hosszú cím1}` vagy
`\MEchapter[r˝Ovid cím2]{hosszú cím2}`
attól függően, hogy `\UseSecondLanguage` definiált vagy sem.
Teljesen hasonló a kisebb formázási egységek használata is.

2 A dokumentum tagolása

- Dokumentum egységek
- Kétnyelvű forráskódok
- Fejezet illusztráció
- Nyomtatott és vetített szöveg összehangolása
- Lebegő objektumok

Néhány könyv stílus lehetővé teszi, hogy a fejezetek elején egy illusztrációt helyezzünk el.

Használata:

`\MEchapterillustration{file}`

Dia készítéskor, a kép egy 'plain' dián jelenik meg. Nyomtatható változatban a fejezet elején jelenik meg a kép.

Ha a fájl név üres, a csomag a `fig/DefaultIllustration.png` képet keresi. Ha a fájl nem található, nem készül illusztráció.

Ha definiáljuk a `\DisableChapterIllustration` makrót, a csomag nem generál képet.

2 A dokumentum tagolása

- Dokumentum egységek
- Kétnyelvű forráskódok
- Fejezet illusztráció
- Nyomtatott és vetített szöveg összehangolása
- Lebegő objektumok

A nyomtatott anyag jelentősen több szöveget szokott tartalmazni, mint a diák. Ezt az extra szöveget úgy lehet a forrás fájlban elhelyezni, hogy az `\ao{text}` (article only) makró belsejében adjuk meg az extra szöveget. Az így megadott szöveg csak a nyomtatott változatban látható, a diákon nem jelenik meg. Vigyázzunk rá, hogy a szöveg mindkét változatban értelmes legyen, különösen, ha mondat belsejében használjuk.

2 A dokumentum tagolása

- Dokumentum egységek
- Kétnyelvű forráskódok
- Fejezet illusztráció
- Nyomtatott és vetített szöveg összehangolása
- Lebegő objektumok

A LaTeX bizonyos objektumokat, úgymint ábrákat, táblázatokat, programlistákat, stb. ún lebegő objektumként kezelhet, tehát nem feltétlenül a forrásnyelvi helynek megfelelő helyen jelennek meg a nyomtatott változatban, viszont a dia képeken igen. Ezért a nyomtatott változatban nem érdemes 'A következő programlistán' módon hivatkozni. Helyette az '`\Aref{lst:hello.cpp} programlista`' mód javasolt.
A dia képeken viszont a megfelelő helyen van a lista, de nincs száma. Ezért az '`A \ao{\ref{lst:hello.cpp}} programlista`' mód az igazi.

Programozás tanításakor alapvető követelmény programlisták megjelenítése. A 'listings' csomag felhasználásával a MultEdu ezt jó minőségben tudja biztosítani. Az itt nem ismertetett részletekért lásd a 'listings' csomag leírását.

Ebben a szakaszban szokatlanul sok elhelyezendő programlista van, ami nagyon megnehezíti a fordítóprogram dolgát. Valódi szövegek esetén az készített oldal sokkal esztétikusabb.

3 Program listák készítése

A megjelenítés beállítása

Sorközi töredék megjelenítése

Teljes programlista megjelenítése

Programlista díszítései

Kapcsolódó egyéb makrók

További program nyelvek

A 'listings' csomag számos lehetőséget biztosít arra, hogy a programlista megjelenítés stílusát ízlésünknek (és a követelményeknek) megfelelően állítsuk be. A MultEdu beállít valamilyen alap-stílust, amit tetszés szerinti alkalommal és módon módosíthatunk.

`\MESetStandardListingFormat`
beállít egy alap-megjelenítést, de nem állít be programnyelvet.

`\MESetListingFormat[options]{language}`
beállítja a nyelvet, és

`\MESetStandardListingFormat`
szerinti alap-megjelenítést és 'options' használatával lehetővé teszi a 'listings' alapértelmezett argumentumainak felülírását.

Gyakori feladat egy rövidebb töredék, mint egyetlen sor vagy akár kulcsszó/változó megjelenítése. Ezt a `\lstinline|code|` módon tehetjük meg.

Az ebben a leírásban is kiterjedten használt LaTeX parancsok megjelenítéséhez a fejezet elején használok egy

```
\MESetListingFormat{TeX}
\lstset{basicstyle= \ttfamily\color{black}\normalsize}
```

vagy

```
\MESetListingFormat[basicstyle=
\ttfamily\color{black}\normalsize]{TeX}
```

parancsot. (különben túl kicsi lesz a megjelenített program kód karaktereinek mérete)

3 Program listák készítése

A `\MESourceFile[keys] {filename} {caption} {label}{scale}` makróval jeleníthetők meg programlisták. Lehetséges kulcs: `wide[=false],decorations[={}]`. A programlista megjelenítéshez használt programsor: `\MESourceFile[language={[ISO]C++}] {lst/HelloWorld.cpp} {A "Hello World"- C++ program} {lst:hello.cpp}{}`

A "Hello World"- C++ program

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{
  // print welcome message
  cout << "Hello World" << endl;
```

Sokszor van szükség szélesebb programlista megjelenítésére. Ennek hatására a két oszlopos nyomtatás teljes szélességében jelenik meg a lista. Egyoszlopos megjelenítés esetén a keskeny lista az oldalszélesség 70%-ára terjed ki, a széles pedig a teljes oldal szélességet igénybe veszi. A széles programlistákat még nehezebb elhelyezni az oldalon (a megjelenítő utasítás helye utáni oldal tetejére kerülhet legelőször), ráadásul nem is szabad felcserélni a normál és széles programlisták sorrendjét. Emiatt a megjelenési hely eléggé messze is kerülhet a hivatkozás helyétől.

A "Hello World"- C++ program, wide

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{
  // print welcome message
  cout << "Hello World" << endl;
  return 0;
}
```

A programlista megjelenítéséhez használt programsor:

```
\MESourceFile[language={[ISO]C++},wide]
{lst/HelloWorld.cpp} {A "Hello World"- C++ program, wide}
{lst:Whello.cpp}{}
```

A programlistán különféle díszítményeket helyezhetünk el. Ehhez a programlista készítésekor használnunk kell a `decorations` kulcsszót és annak argumentumaként az e szakaszban bemutatott makrókat kell megadni.

Az általános forma:

```
\MESourceFile[options, decorations={ list of decorations }
] {source file} {caption} {label}{}
```

ahol a díszítések listája a szakaszban felsorolt bármelyik fajta díszítést tartalmazhatja. Az `options` argumentumaként a 'listings' csomagban használt bármely opció használható.

A programlistán a programtörzs utasításainak kijelöléséhez a
`\MESourceFile[language={[ISO]C++}, decorations={`
`\MESourcelinesHighligh {HelloBalloon} {lst:HLhello.cpp}`
`{6}{8} } ] {lst/HelloWorld.cpp} {"Hello World" -- a C++`
`way, kijel´Olt} {lst:HLhello.cpp}{}`
parancsot kell kiadni.

"Hello World" – a C++ way, kijelölt

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{
  // print welcome message
  cout << "Hello World" << endl;
  return 0;
```

Az előbbi programlistán a kijelöléshez megjegyzést is fűzhetünk. Ennek formája

`\MEBalloonComment[keys]{BallonName} {ShiftPosition} {Comment} {CommentShape}`

amivel az előzőleg felrajzolt ballonhoz fűzhetünk megjegyzést. Itt `BallonName` az `\MEHighlightLines` első argumentuma, `ShiftPosition` a megjegyzésdoboz eltolása, `Comment` pedig maga a megjegyzés. A lehetséges opciók: `width[=3cm]` és `color[=deeppeach]`.

A programlista készítéséhez a

```
\MESourceFile[language={[ISO]C++},wide, decorations={
\MESourcelinesHighlight {HelloBalloon} {lst:HLChello.cpp}
{6}{8} \MESourceBalloonComment {HelloCBalloon} {0cm,0cm}
{This is the body} {CommentShape} } ] {lst/HelloWorld.cpp}
{"Hello World" -- egy C++ program } {lst:HLhello.cpp}{}
```

parancsot kell kiadni.

"Hello World" – egy C++ program

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{
  // print welcome message
  cout << "Hello World" << endl;
```

Ez a csoport

Az egyes forráskód sorokhoz is fűzhetünk megjegyzéseket, lásd programlista. Ehhez a

```
\MESourceFile[language={[ISO]C++}, decorations={
\MESourcelineComment{lst:Chello.cpp} {6} {-1cm,0cm} {This
is a comment} {CommentShape} } ]{lst/HelloWorld.cpp}
{"Hello World" -- a C++ way, commenting source lines}
{lst:Chello.cpp}{}
```

utasítást kellett kiadni.

"Hello World" – a C++ way, megjegyzés a forráskód sorához

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{
  // print welcome message
  cout << "Hello World" << endl;
  return 0;
```

Ez megjegyzés

Az előbbi programlistán különböző programsorokat is megjelölhetünk. Ennek formája

`\MESourceListBalls[keys]{ListingLabel}{List of lines}`

amivel a megjelölt programsorok végére kerül egy-egy számozott golyó. Itt `ListingLabel` a programlista címkéje, `List of lines` pedig azon sorszámok listája, ahová golyót szeretnénk elhelyezni. Lehetséges kulcsok, az alapértelmezett értékkel:

`color[=orange]` and `number[=1]`.

Megjegyzések:

- Dia készítéskor az egyes golyók a egy dia sorozatra kerülnek
- A golyók elhelyezése csak geometria pozíció alapján történik, nem veszi figyelembe a 'firstline' paramétert.
- a golyók számozása a `number[=1]` értéktől indul.

Az így megjelölt sorokra később így hivatkozhatunk: "(2) a programtörzset lezáró visszatérési utasítás". Ehhez a `\MEBall{lst:LBhello.cpp}{2}` makrót kell használnunk.

A lista készítéséhez a
`\MESourceFile[language={[ISO]C++}, decorations={ \MESourcelineListBalls{lst:LBhello.cpp}{3,8,5} } ] {lst/HelloWorld.cpp} {"Hello World" -- a C++ way, golyokkal} {lst:LBhello.cpp}{}`
parancsot kell kiadni.

"Hello World" – a C++ way, golyókkal

A lista készítéséhez a
`\MESourceFile[language={[ISO]C++}, decorations={ \MESourcelineListBalls{lst:LBhello.cpp}{3,8,5} } ] {lst/HelloWorld.cpp} {"Hello World" -- a C++ way, golyokkal} {lst:LBhello.cpp}{}`
parancsot kell kiadni.

"Hello World" – a C++ way, golyókkal

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{
  // print welcome message
  cout << "Hello World" << endl;
  return 0; 2
```

A lista készítéséhez a
`\MESourceFile[language={[ISO]C++}, decorations={ \MESourcelineListBalls{lst:LBhello.cpp}{3,8,5} } ] {lst/HelloWorld.cpp} {"Hello World" -- a C++ way, golyokkal} {lst:LBhello.cpp}{}`
parancsot kell kiadni.

"Hello World" – a C++ way, golyókkal

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{ 3
  // print welcome message
  cout << "Hello World" << endl;
  return 0;
```

Néha ábrát is akarhatunk elhelyezni a programlistán. Az ezt a célt szolgáló makró

`\MESourcelineFigure[keys] {SourceLabel} {LineNo} {ShiftPosition} {GraphicsFile}`.

Lehetséges kulcs: `width[=3cm]`

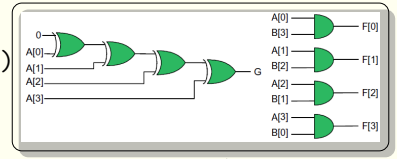
A programlista előállításához használt makró:

```
\MESourceFile[language={Verilog},wide, decorations={
\MESourcelineFigure[width=5.2cm] {lst:forloops.v}{8}
{3.0,-.3} {fig/forloops} } ] {lst/forloops.v}
{Implementing \lstinline|for| loop with repeating HW}
{lst:forloops.v}{}
```

'for' ciklus megvalósítása HW ismétléssel

```
// for == repeat HW

always @(A or B)
begin
  G = 0;
  for (I = 0; I < 4; I = I + 1)
  begin
    F[I] = A[I] & B[3-I];
    G = G ^ A[I];
```



Néha érdemes forrás kód fájlokat egymás mellé helyezve összehasonlítani. Az erre szolgáló makró

```
\MESourceFileCompare[keys]{source file1} {source file2}
{caption} {label}
```

A programlista előállításához használt utasítás

```
\MESourceFileCompare[language={[ANSI]C}] {lst/lower1.c}
{lst/lower2.c} {Comparing two routines for converting
string to lower case} {lst:lower12.c}
```

A két kisbetűssé alakító rutin összehasonlítása

```c
/* Convert string to lowercase: slow */
void lower1(char *s)
{
   int i;

   for (i = 0; i < strlen(s); i++)
   if (s[i] >= 'A' && s[i] <= 'Z')
    s[i] -= ('A' - 'a');
}
```

```c
/* Convert string to lowercase: faster */
void lower2(char *s)
{
  int i;
  int len = strlen(s);
  for (i = 0; i < len; i++)

    if (s[i] >= 'A' && s[i] <= 'Z')
      s[i] -= ('A' - 'a');
}
```

A makró a forrásfájlt nem kezeli; az ábrán a jobb összehasonlítás kedvéért beiktatott üres sorokat kézzel kellett beírni.

A program lista a
`\MESourceFileWithResult[language=C++,wide, decorations={\MESourcelineListBalls {lst:calculatorwithresult} {13,14,16,18,19} }] {lst/expensive_calculator.cpp} {lst/calculatorresult.txt} {The calculator program with its result} {lst:calculatorwithresult}`
utasítás eredménye.
Néha hasznos egy forrásfájlt a futtatás eredményével együtt megmutatni. A
`\MESourceFileWithResult[keys]{source file} {result file} {caption} {label}`
makró ezt teszi lehetővé. A forráskódban it is megjelölhetünk 'nevezetes pontokat', az eredményfájlban ez nem lehetséges.

A kalkulator program és eredménye

```
// Expensive Calculator
// Demonstrates built-in arithmetic operators

#include <iostream>
using namespace std;

int main()
{
  cout << "7 + 3 = " << 7 + 3 << endl;
  cout << "7 - 3 = " << 7 - 3 << endl;
  cout << "7 * 3 = " << 7 * 3 << endl;

  cout << "7 / 3 = " << 7 / 3 << endl;
  cout << "7.0 / 3.0 = " << 7.0 / 3.0 << endl;

  cout << "7 % 3 = " << 7 % 3 << endl;

  cout << "7 + 3 * 5 = " << 7 + 3 * 5 << endl;
  cout << "(7 + 3) * 5 = " << (7 + 3) * 5 << endl;

  return 0;
}
```


```
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2
7.0 / 3.0 = 2.33333
7 % 3 = 1
7 + 3 * 5 = 22
(7 + 3) * 5 = 50
```

A kalkulator program és eredménye

```
// Expensive Calculator
// Demonstrates built-in arithmetic operators

#include <iostream>
using namespace std;

int main()
{
  cout << "7 + 3 = " << 7 + 3 << endl;
  cout << "7 - 3 = " << 7 - 3 << endl;
  cout << "7 * 3 = " << 7 * 3 << endl;

  cout << "7 / 3 = " << 7 / 3 << endl;
  cout << "7.0 / 3.0 = " << 7.0 / 3.0 << endl;

  cout << "7 % 3 = " << 7 % 3 << endl;

  cout << "7 + 3 * 5 = " << 7 + 3 * 5 << endl;
  cout << "(7 + 3) * 5 = " << (7 + 3) * 5 << endl;

  return 0;
}
```


```
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2
7.0 / 3.0 = 2.33333
7 % 3 = 1
7 + 3 * 5 = 22
(7 + 3) * 5 = 50
```

A kalkulator program és eredménye

```
// Expensive Calculator
// Demonstrates built-in arithmetic operators

#include <iostream>
using namespace std;

int main()
{
  cout << "7 + 3 = " << 7 + 3 << endl;
  cout << "7 - 3 = " << 7 - 3 << endl;
  cout << "7 * 3 = " << 7 * 3 << endl;

  cout << "7 / 3 = " << 7 / 3 << endl;
  cout << "7.0 / 3.0 = " << 7.0 / 3.0 << endl;

  cout << "7 % 3 = " << 7 % 3 << endl;

  cout << "7 + 3 * 5 = " << 7 + 3 * 5 << endl;
  cout << "(7 + 3) * 5 = " << (7 + 3) * 5 << endl;

  return 0;
}
```


```
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2
7.0 / 3.0 = 2.33333
7 % 3 = 1
7 + 3 * 5 = 22
(7 + 3) * 5 = 50
```

A kalkulator program és eredménye

```
// Expensive Calculator
// Demonstrates built-in arithmetic operators

#include <iostream>
using namespace std;

int main()
{
  cout << "7 + 3 = " << 7 + 3 << endl;
  cout << "7 - 3 = " << 7 - 3 << endl;
  cout << "7 * 3 = " << 7 * 3 << endl;

  cout << "7 / 3 = " << 7 / 3 << endl;
  cout << "7.0 / 3.0 = " << 7.0 / 3.0 << endl;

  cout << "7 % 3 = " << 7 % 3 << endl;

  cout << "7 + 3 * 5 = " << 7 + 3 * 5 << endl;
  cout << "(7 + 3) * 5 = " << (7 + 3) * 5 << endl;

  return 0;
}
```

4

```
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2
7.0 / 3.0 = 2.33333
7 % 3 = 1
7 + 3 * 5 = 22
(7 + 3) * 5 = 50
```

A kalkulator program és eredménye

```
// Expensive Calculator
// Demonstrates built-in arithmetic operators

#include <iostream>
using namespace std;

int main()
{
  cout << "7 + 3 = " << 7 + 3 << endl;
  cout << "7 - 3 = " << 7 - 3 << endl;
  cout << "7 * 3 = " << 7 * 3 << endl;

  cout << "7 / 3 = " << 7 / 3 << endl;
  cout << "7.0 / 3.0 = " << 7.0 / 3.0 << endl;

  cout << "7 % 3 = " << 7 % 3 << endl;

  cout << "7 + 3 * 5 = " << 7 + 3 * 5 << endl;
  cout << "(7 + 3) * 5 = " << (7 + 3) * 5 << endl;

  return 0;
}
```

5

```
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2
7.0 / 3.0 = 2.33333
7 % 3 = 1
7 + 3 * 5 = 22
(7 + 3) * 5 = 50
```

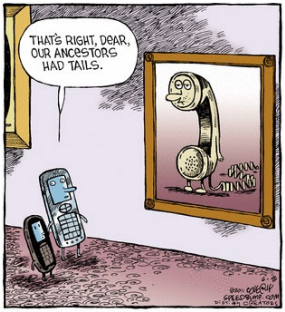
3 Program listák készítése

Saját céljaimra a 'listings' csomagban definiáltakon felül, további program nyelveket definiáltam:

- diff
- [DIY]Assembler
- [ARM]Assembler
- [x64]Assembler
- [y86]Assembler

4 Ábrák beszúrása

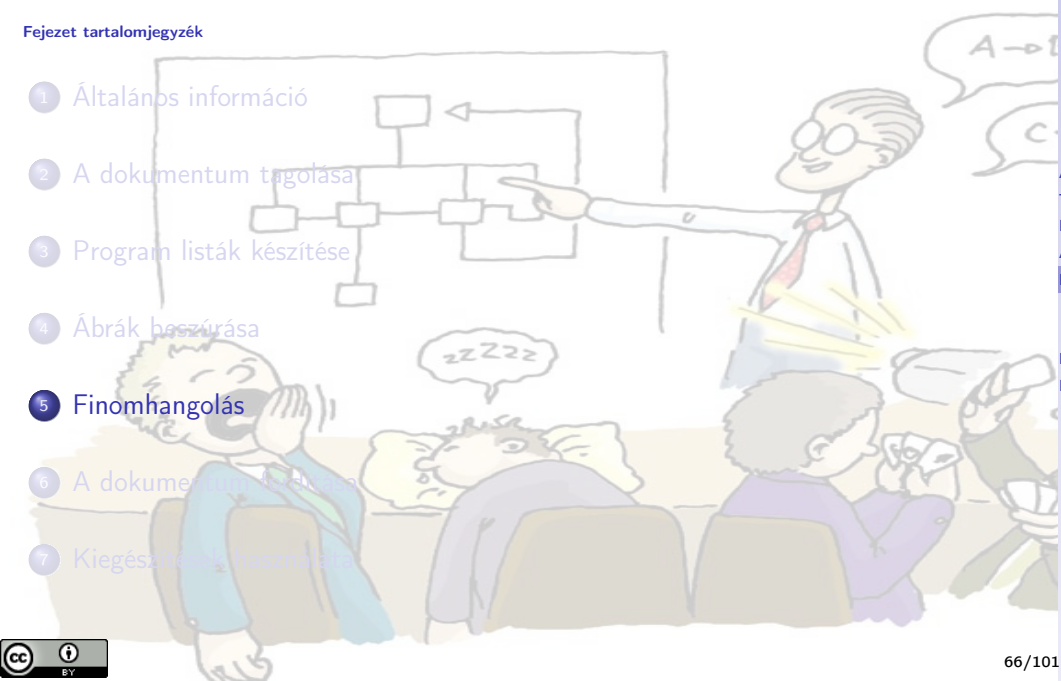
Hagyományos ábrák



Amikor régi és új telefonok találkoznak

Az ábra előállításához a

```
\MEfigure{fig/phone_anchestors} {{Regi es uj telefonok ha
talalkoznak}} {fig:phonenachestors} {2011
http://pinterest.com}{.8}
```



A MultEdu rendszer tökéletesen működik alapértelmezett beállításokkal is, de nem gondolatolvasó. A beállításokat `\def{\xxx}` formájú definíciókkal lehet megváltoztatni. A beállítások helye üzemmódtól függ, a részleteket lásd a 6 szakaszban. Az alapértelmezett beállítások az egyes beállítások hatásának részletes leírásánál találhatók. A fejezet következő szakaszai az üzemmódok használatát mutatja be.

5 Finomhangolás

Alap beállítások

A `MultEdu` csomag beállítási lehetőségei

A `MultEdu` csomag fájljai

A MultEdu beállítási lehetőségként vagy fájlok megadott helyen és néven való előfordulását, vagy pedig \def{Option{Value}} formájú definiciók előfordulását tudja értelmezni. Ezek hiánya esetén az alapértelmezett viselkedés lét életbe az eredmény fájl előállítása során.

A beállítási lehetőségek lehetnek kötelezően használandók; az eredmény fájlt nagy mértékben befolyásolók, vagy csak kisebb finomítást jelentők; csak bizonyos típusú eredmény fájl készítésekor hatásosak.

A MultEdu lehetővé teszi kétféle elterjedt formátum használatát. Egyre gyakoribb 16:9 arányú képformátum így az az alap beállítás. A 4:3 arányú képformátumot a

```
{\def\DisableWideScreen{YES}}
```

definiálásával lehet beállítani.

Néha (főként rövid bemutatók esetén) egyáltalán nincs szükség tartalomjegyzékre. Ezt a

```
{\def\DisableTOC{YES}}
```

definiálásával lehet elérni. Az is előfordul, hogy a fejezet-szintű tartalomjegyzék még szükséges, de a szakasz szintű már nem. Ezt a

```
{\def\DisableSectionTOC{YES}}
```

definiálásával lehet elérni.

A használt fájloknak illeszkedni kell a fájlok általános rendszerébe, lásd 2 szakasz. Tanácsos csak a projekt könyvtárba tartozó fájlokat változtatni, mivel a csomag közösen használt fájljai a kötegelt feldolgozás során felülíródnak.

A dokumentumokhoz tartozik néhány fejzet leíró definíció. Mintaként a felhasználói leírás `src/Heading.tex` fájlja szolgál.

A fejzet tartalma:
A `\def\LectureAuthor{V\'egh J\'anos}` sor adja meg a szerzőt, a `\def\LectureTitle{Hogyan haszn\'aljuk\\ a MultEdu csomagot}` a címét, a `\def\LectureSubtitle{(Hogyan k\'esz\'\i{}ts\"unk \'erdekes\\ \'es vonz\'o tananyagot)}` pedig a dokumentum címét és alcímét. Megadhatunk egy `\def\LecturePublisher{Egyetem neve vagy konferencia neve}` meghatározást is. Javasolt egy `\def\LectureRevision{V\Version\ \at year.mm.dd}` formájú sor használata is

Kétnyelvű dokumentumok készítéséhez a fentieket

```
\ifthenelse{\equal{\LectureLanguage}{magyar}}
{% in Hungarian
}% true
{% NOT magyar
}
```

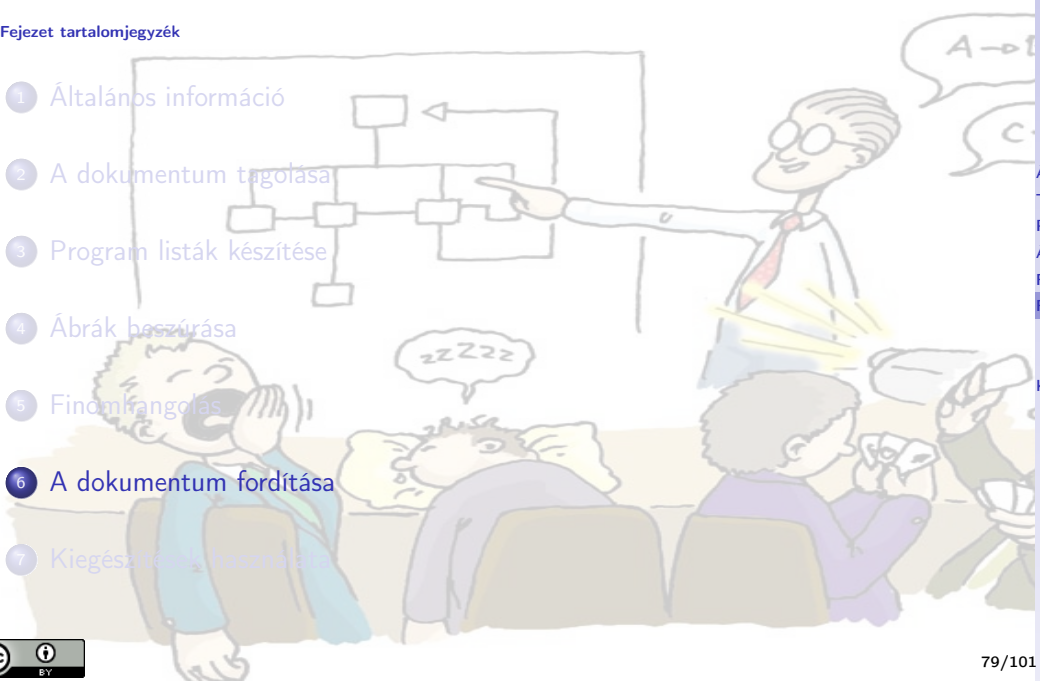
blokkban kell elhelyezni.

Megadhatunk számítógépes címet is

```
\def\LectureEmail{Janos.Vegh\at unideb.hu}
```

Ugyancsak itt célszerű megadni a dokumentumban használt BibTeX fájlokat, akár a nyelv, vagy a fájl tényleges fellelhetősége alapján:

```
\IfFileExists{src/Bibliographyhu}
{\def\LectureBibliography{src/Bibliography,
src/Bibliographyhu}}
{\def\LectureBibliography{src/Bibliography}}
```

A `Main.tex` fájl a közös és a két fordítási módban egyformán használt rész: ez tartalmazza a tényleges forráskódot. Az ebben a fájlban (továbbá az ide beolvasott fájlokban) szereplő bármely beállítás változtatás megváltoztatja a rendszer beállításait, azaz itt nem tanácsos bármiféle beállítást használni. Érdemes az összes beállítást egyetlen fájlba gyűjteni, amit aztán a fő fájl magába olvas.

A tananyag fejlesztést általában valamilyen szerkesztőbe integrált fejlesztő rendszerrel érdemes végezni. A szerkesztőbe be kell olvasni a boríték fájlt (a `Demo.tex` megfelelőjét) és azt gyökér dokumentumként megjelölni. A `Main.tex` fájlban érdemes hozzáadni a hivatkozásokat a tananyag fejezeteire, ami anyagokat természetesen a `src` alkönyvtárban célszerű elhelyezni, követve a demonstrációs anyag elrendezését.

A beállítások tárolására szolgáló fájlt is a `src` alkönyvtárban érdemes elhelyezni, célszerűen `Defines.tex` néven. A burkolóként szolgáló `Demo.tex` feladata, hogy ezt és a fő fájlt beolvassa.

A kötegelt mód a konfigurálás során készít egy `Defines.tex` fájlt, de az a `build/build/src` alkönyvtárba kerül. (Onnét lehet puskázni, hogy mit és hogyan érdemes beállítani; miután egyszer már futott a kötegelt fordítás.)

A kötegelt fordítás egy "minta" fájlt is készít `Defines.tex.in` néven a `src` alkönyvtárba. Ennek a két fájlnak a tartalma a kötegelt fordítás utolsó menetének felel meg.

A kötegelt fordítás (főként) arra szolgál, hogy a közös forráskódból kényelmesen tudjuk előállítani a különféle formátumokban és nyelveken anyagainkat.

Technikai okokból a tényleges fordítás előtt a rendszer saját másolatot készít a MultEdu szükséges fájljairól a projekt `common` alkönyvtárába. Ezzel a saját kópiával lehet kísérletezni, vagy akár törölni; a következő kötegelt fordítás majd helyreállítja. (azaz a következő fordítás előtt az értékes fejlesztést el kell menteni, akár a `../../common` alkönyvtárba, ha azt másutt is használni akarjuk.)

A fordítás három lépésből áll.

- a projekt könyvtárban a `CMakeLists.txt` fájlban be kell állítani az adott fordításban használni kívánt beállításokat
- a projekt `build/build` alkönyvtárára váltani, majd kiadni a `cmake ../..` parancsot.
- ugyanitt adjuk ki a `make` parancsot, aminek hatására a tényleges fordítás elindul.

A MultEdu alap-beállításait \def{OptionName} utasításokkal lehet meghatározni. Amennyiben a fordítás előtt a fordítóprogram nem talál ilyen meghatározást, az alapbeállítást használja. A kézi és a kötegelt fordítás beállításai különböznek. A kötegelt feldolgozás esetén a fordítóprogram a CMakeFiles.txt fájlban megadott beállításokkal újonnan létrehozott build/build/src/Defines.tex meghatározásokat használja, a kézi fordítás pedig a src/Defines.tex meghatározásokat. Ezek célszerűen megegyeznek, de az utóbbi beállításokat a felhasználónak kell megadni.

A MultEdu a standard háromszintű verzió számozást használja (fő és alszám, valamint folt). A MultEduval készült anyagoknak kétféle verziója van: a saját tananyagának verzióját a felhasználó tartja karban, a MultEdu változatát pedig a fejlesztő.
A MultEdu verziószáma a `../../common/MEMacros.tex` fájlban található; célszerű változatlanul hagyni. A saját kurzus anyag verzióját a `CMakeFiles.txt` file tartalmazza, az minden kötegelt fordítás alkalmával frissül a `Defines.tex` fájlban. A kézi fordításnak saját beállításai vannak, de célszerű azt átvenni a generált fájlból.
A saját verzió száma a generált kimeneti fájl nevében is szerepel, tehát érdemes következetesen használni azt. Használata:

```
\def\Version{nagy.kis.folt}
```

A MultEdu egy- és két-nyelvű dokumentumokat tud kezelni. A különböző nyelvekhez különböző helyesírás, fejezetcímek, feliratok tartoznak. A beállításoknál kell megadni a nyelvet: ezt pl. a `\LectureLanguage{magyar}` beállítással lehet megtenni (enélkül az alapbeállítás `\LectureLanguage{english}`).
A kiválasztott nyelv neve az eredmény file nevében is megjelenik.

A kétnyelvű dokumentumokban van egy első és egy második nyelv, amilyen sorrendben szerepelnek a nyelvi szövegek a dokumentumban. Ez lehetővé teszi, hogy az egymás alatt levő kétféle nyelvű kurzus anyagot összhangban tudjuk fejleszteni. A nyelv kiválasztásával a két anyag bármelyikéből tudjunk eredmény fájlt generálni. Ha a `\UseSecondLanguage{}` definiálva van, a sorrendben második nyelvet fogja a csomag feldolgozni, és arra a `\LectureLanguage{}` által megadott szabályokat használja.
Kötegelt fordítás esetén meg kell adnunk a `FirstLanguage` és `SecondLanguage` értékét (azaz, hogy az elsőként és másodikként megtalált szöveg milyen nyelvű). Ha bekapcsoljuk a `NEED_BOTH_LANGUAGES` kapcsolót, a kötegelt feldolgozás során mindkét nyelvű kimenő fájt előállítja a rendszer. Ha ez ki van kapcsolva, akkor a `USE_SECOND_LANGUAGE` kapcsoló dönti el, melyik nyelvet fogja a rendszer használni.

Különösen technikai jellegű tárgyak esetén, gyakran szerepelnek rövidítések, betűszavak, illetve bizonyos fogalmak egyértelmű meghatározása. A MultEdu a `glossaries` csomagot használva teszi lehetővé, hogy a dokumentumokban ilyeneket használjon, ráadásul hiper-hivatkozásként. Az ilyen elemeket a szövegben a `\gls{ref}` módon kell elhelyezni. A szövegben ennek hatására megjelenik az elem rövid neve, és annak első előfordulásakor annak rövid leírása is. Bővebben lásd a `glossaries` csomag leírását.

Különösen technikai jellegű tárgyak esetén, gyakran szerepelnek rövidítések, betűszavak, illetve bizonyos fogalmak egyértelmű meghatározása. A MultEdu a `glossaries` csomagot használva teszi lehetővé, hogy a dokumentumokban ilyeneket használjon, ráadásul hiper-hivatkozásként. Az ilyen elemeket a szövegben a `\gls{ref}` hivatkozásként kell elhelyezni, és a nyomtatott szövegben ennek hatására azon a helyen az elem rövid neve jelenik meg, és a rövidítések feloldására, a hivatkozás első előfordulásakor annak rövid leírása is. Bővebben lásd a `glossaries` csomag leírását.

Ha mintaként használja a számítógép fogalmát, ahol Central Processing Unit, központi egység (CPU) valamint Direct Memory Access, közvetlen memória elérés (DMA) is előfordul akkor a szövegben a

```
Ha \gls{minta}k\'ent haszn\'alja a \gls{szamitogep}
fogalm\'at, ahol \gls{CPU} valamint \gls{DMA} is
el\H{o}fordul
```

módon kell azt használni. Ilyenkor a MultEdu hozzáfűzi a dokumentumhoz a `Acronyms` and `Glossary` fejezeteket, ahol a megjelölt hivatkozások kifejtése található. A dokumentum olvasásakor a hivatkozásra kattintva, az olvasó program a kifejtésre ugrik, ahonnét az oldalszámra kattintva, folytathatja az olvasást.

A MultEdu azt várja, hogy (ha használni akar ilyen lehetőséget) a projekt tartalmaz egy `src/Glossary.tex` fájlt, ahol a hivatkozások részletes kifejtése megtalálható. A bemutatott mintában a bejegyzések kódolása:

```
\ifthenelse{\equal{\LectureLanguage}{english}}
{
\newglossaryentry{computer}
{
name={computer},
description={is a programmable machine that receives input,
stores and manipulates data, and provides
output in a useful format}
}
\newglossaryentry{sampleone}{name={sample},description={a
little example}}
\newacronym{CPU}{CPU}{Central Processing Unit}
\newacronym{DMA}{DMA}{Direct Memory Access}
}
{}
```

Ezeknek a lehetőségeknek csak a nyomtatható változatok esetén van szerepe. A `beamer` alapú formátumok nem generálnak ilyen jegyzékeket, de a `\gls{ref}` természetesen ott is használható.
Nagyon jó lehetőség arra, hogy a rövidítés kifejtés, fogalom magyarázat, stb. ne törje meg a szöveget, de azért mindig kéznél legyen.