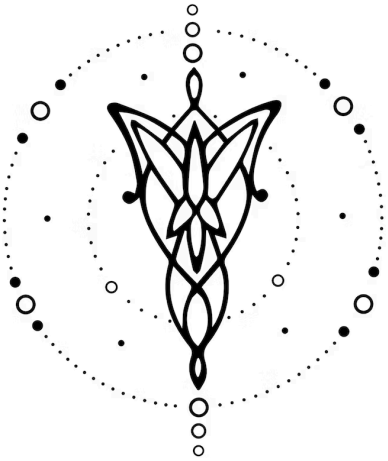

LangTeX

Entrega Final

v1.0.0

Integrantes

Nombre	Apellido	Legajo	E-mail
Lautaro	Bonseñor	62842	lbonsenor@itba.edu.ar
Camila	Lee	63382	clee@itba.edu.ar
Fernando	Li	64382	feli@itba.edu.ar
Matías	Rinaldo	60357	mrinaldo@itba.edu.ar

Repositorio

https://github.com/lbonsenor/tpe-tla

1. Introducción

1.1 Motivación y Contexto

El aprendizaje de idiomas presenta desafíos únicos en la documentación académica, especialmente cuando se requiere combinar múltiples sistemas de escritura, romanizaciones y elementos interactivos en un mismo documento. LaTeX, siendo el estándar de facto para documentos académicos, ofrece capacidades tipográficas excepcionales, pero su sintaxis compleja puede resultar intimidante para educadores y estudiantes de idiomas que no poseen experiencia técnica profunda.

La creación de materiales didácticos multilingües tradicionalmente requiere el dominio de numerosos paquetes LaTeX especializados, configuraciones complejas de fuentes CJK (chino, japonés, coreano), y el manejo manual de sistemas de romanización. Esta complejidad técnica frecuentemente limita la capacidad de los educadores para crear contenido didáctico rico y accesible.

1.2 Propuesta de Solución

LangTeX surge como respuesta a esta problemática, proponiendo un Domain Specific Language (DSL) que actúa como una capa de abstracción sobre LaTeX, específicamente diseñada para la creación de materiales educativos multilingües. El lenguaje adopta una filosofía similar a Markdown en términos de simplicidad sintáctica, pero mantiene toda la potencia tipográfica de LaTeX.

La propuesta central consiste en introducir comandos especializados con sintaxis [!comando] que encapsulan la complejidad de LaTeX mientras proporcionan funcionalidades específicas para el dominio educativo multilingüe, incluyendo:

- **Traducción automática con romanización**: Soporte nativo para múltiples sistemas de escritura con romanización automática
- **Estructuras didácticas especializadas**: Ejercicios interactivos, diálogos, y tablas comparativas
- **Compatibilidad total con LaTeX**: Permitiendo la integración gradual en flujos de trabajo existentes

2. Modelo Computacional

2.1. Lenguaje

El dominio de este lenguaje está definido en la optimización y simplificación de la escritura en LaTeX para la toma de notas, orientado al aprendizaje de idiomas, permitiendo que los usuarios trabajen con notas y documentos de manera más eficiente sin comprometer las ventajas que trae el mismo.

Basándose en la idea propuesta, el lenguaje va a ser de tipo DSL (Domain Specific Language), pues el lenguaje se especifica en un único dominio, ya que simplifica la toma de notas dentro del mismo ecosistema de LaTeX, el cual se centra en la redacción y formateo de textos académicos.

2.2. Lenguaje

2.2.1. Conceptos Lingüísticos y su Representación

Concepto del Dominio	Representación en LaNgTeX	Ejemplo
Carácter/palabra con anotaciones	Tipo `Translation`	`[!translate](lang="ko"){안녕하세요}{hola}`
Bloque de texto en idioma extranjero	Tipo `Block`	`[!block]{this is a block}`

Comparativa entre idiomas	Tipo `Table`	`[!table](cols=2){`
		`  [!row]{header=true}{header1}{header2}`
		`  [!row]{col1}{col2}`
		`  [!row]{col1}{col2}`
		`}`

Sistema de ejercicios	Tipo `Exercise`	`[!exercise](type= "multiple-choice"){`
		`  [!prompt]{El profesor [!fill] dio un 10.}`
		`  [!options]{nos}{les}{le}`
		`  [!answer]{1}`
		`}`

Diálogos para practicar idiomas	Tipo `Dialog`	`[!dialog]{`
		`  [!speaker](name="John"){Hi, Mary}`
		`  [!speaker)(name="Mary"){Heyo, John}`
		`}`

2.2.2. Principios de Composición

El lenguaje está diseñado con principios de composición que permiten:

1. **Anidamiento controlado**: Comandos específicos pueden contener otros comandos según reglas semánticas definidas

2. **Extensibilidad paramétric**a: Cada comando acepta parámetros opcionales para personalización
3. **Compatibilidad con LaTeX**: Texto LaTeX puro puede coexistir naturalmente con comandos LangTeX

3. Implementación

3.1. Frontend

3.1.1. Analizador Léxico

Decisiones de Diseño de Contextos

Para LangTeX se implementaron tres contextos que reflejan la estructura jerárquica del lenguaje:

- Contexto INITIAL: Maneja texto LaTeX estándar y la detección de comandos LaNgTeX. La decisión de mantener compatibilidad con LaTeX permitió reutilizar editores existentes con syntax highlighting básico.
- Contexto LANGTEX: Se activa tras reconocer comandos cómo [!translate]. Esta separación fue crucial para distinguir entre los comandos de Latex y los nuestros.
- Contexto PARAM: Dedicado exclusivamente al análisis de parámetros dentro de los comandos LaNgTeX. Esta decisión simplifica significativamente el manejo de strings con espacios, valores booleanos y números enteros dentro de los parámetros.

```
"[!translate]"    { BEGIN(LANGTEX); return TRANSLATE_COMMAND; }
<LANGTEX>\(       { BEGIN(PARAM); return OPEN_PARENTHESIS; }
<PARAM>\)         { BEGIN(LANGTEX); return CLOSE_PARENTHESIS; }
```

Evolución del Manejo de Whitespace

Durante la realización de la tercera entrega, orientada principalmente en el desarrollo del Backend, se descubrió la obligación de no poder ignorar saltos de línea, sin embargo, esto trajo problemas a la hora de análisis sintáctico, pues comandos tanto LaTeX como LaNgTeX necesitaban un formato muy estricto y específico (sin permitir saltos de línea después de las llaves).

```
[!dialog](title="Greeting"){
    [!speaker](name="John"){Hi Mary}
    [!speaker](name="Mary"){Hi John}
}
```

Figura 1. Antes del cambio

A partir de esto, pensando en lenguajes como Python, que obliga al usuario a indentar su código, se nos ocurrió que el usuario esté obligado a seguir ciertas reglas de estilo, como lo es en

el caso de [!row] y [!table] (un table tiene múltiples rows, sin embargo, cada row está obligado a estar espaciado de otro por un salto de línea).

```
[!dialog](title="Greeting")
{
    [!speaker](name="John"){Hi Mary}
    [!speaker](name="Mary"){Hi John}
}
```

Figura 2. Luego del cambio

A su vez, se decidieron cambiar algunos nombres de parámetros tales como el caso del comando Exercise. En vez de los tipos "fill" y "translate", ahora pasan a nombrarse "multiple-choice" y "single-choice", que resultan ser más intuitivos a lo propuesto (figura 3).

```
[!exercise](type="multiple-choice", title="Ejercicio de completar")
{
   [!prompt]{Completa la frase: 我 [!fill] 吃饭}
   [!options]{想}{有}{去}
   [!answer]{1}
}

[!exercise](type="single-choice", title="Ejercicio de traducción")
{
   [!prompt]{Traduce: Me gusta el cafe}
   [!answer]{我喜欢咖啡}
}
```

Figura 3. Nuevos valores en los parámetros de exercise.

3.2. Backend

3.2.1 Análisis Semántico

Desarrollo del Comando Translate

Durante el desarrollo surgió la necesidad de definir qué comandos LaTeX serían permitidos dentro del contenido de `[!translate]`. Se implementó una validación restrictiva que solo permite comandos de formateo de texto:

```
static const char *ALLOWED_LATEX_COMMANDS[] = {
    "\\textbf",    // Bold text
    "\\textit",    // Italic text
    "\\color",     // Color command
    "\\underline", // Underline
    "\\emph",      // Emphasis
    NULL           // Sentinel
};
```

Esta decisión se tomó para mantener la compatibilidad con la macro \text{} de LaTeX ya que la macro creada para translate está construida con \text. De esta manera se asegura que comandos como \textit (cursiva), \textbf (negrita), y \color{red} funcionen correctamente, mientras se evita recursión compleja que pudiera comprometer la estabilidad del generador.

Validación de Idiomas Soportados

Se implementó verificación durante el análisis semántico para asegurar que los idiomas utilizados sean los permitidos. Se decidió no hacer obligatorio el parámetro `lang` dentro del comando translate, pero esto significa que la romanización simplemente imprime el texto original que se pretendía romanizar.

Integración del Sistema de Romanización

Se desarrolló una integración con un módulo externo de romanización ubicado en una carpeta separada. Se decidió únicamente incluir las siguientes romanizaciones:

- Coreano (Hangul): Para verificar idiomas asiáticos (además, hay dos integrantes que hablan coreano, por lo que se pudo validar mejor la romanización)
- Hebreo: Por su escritura de derecha a izquierda (RTL)
- Ruso (Cirílico): Como representante de alfabetos cirílicos

Validación de Diálogos y Tablas

Para comandos como `[!dialog]` y `[!table]` el analizador semántico se asegura de que no se permitan otros comandos excepto `[!speaker]` (para dialog) y `[!row]` (para table). Esta restricción mantiene la coherencia estructural del código generado.
Se decidió que el comando `[!table]` requiere validación cruzada entre el parámetro ***cols*** y el contenido real de cada fila para asegurar consistencia en la estructura tabular.

Para los ejercicios de opción única y opción múltiple, se permitieron los comandos `[!prompt]`, `[!options]` y `[!answer]`. Se implementó validación específica para asegurar que las respuestas correspondan a opciones válidas en ejercicios de opción múltiple.

Manejo de Parámetros

El propósito principal del analizador semántico fue verificar que los parámetros fueran los apropiados. Se decidió que cuando se inserte un parámetro desconocido, en lugar de lanzar un error, se lance una advertencia y se permita que la compilación continúe:

```
logWarning(_logger, "[!translate] unknown parameter '%s'", current->param->key);
```

Esta decisión prioriza la extensibilidad futura y la usabilidad, permitiendo que el código existente siga funcionando cuando se agreguen nuevos parámetros.

Sistema de Buffering para Contenido Dinámico

Para manejar la generación de contenido que debe ser capturado como string en lugar de enviado directamente al output, se decidió implementar un sistema de buffering dinámico. Este mecanismo redirige temporalmente la salida estándar hacia un buffer interno, permitiendo capturar el contenido generado como variable de tipo char para su posterior reutilización.

El caso principal de uso es la romanización, donde necesitamos el contenido textual como parámetro de entrada para la función de romanización:

```
_start_buffering();
_generateContent(level, command->leftText);
char *left_content = _stop_buffering();

char *romanizedWord = romanize(language, left_content);
_output(level, "\\rom[");
```

```
        _generateContent(level, command->rightText);
        _output(level, "]{%s}{%s}", left_content, romanizedWord);
```

Este sistema permite alternar entre el modo de generación normal (salida directa) y el modo de captura (buffer interno), facilitando el procesamiento de contenido que requiere múltiples pasadas o transformaciones antes de su inclusión en el documento final.

Exclusión del uso de la Tabla de Símbolos

Se consideró que no fue necesario el uso de las tablas de símbolos, ya que en los únicos casos en donde se requerían un chequeo de tipo de datos, son los parámetros de los comandos LaNgTeX. Dado que las últimas ya tienen almacenado en el struct el tipo de dato:

```
        struct LangtexParam {
                char * key;
                union {
                        char * stringParam;
                        int intParam;
                        boolean boolParam;
                } value;
        LangtexParamType type;
};
```

Se consideró más pertinente realizar un chequeo directo (sin hacer uso de tablas). A su vez, como en el presente proyecto no se crean variables/funciones, también fue una razón más por la cuál no se emplearon tablas de símbolos.

3.2.2 Generación de Código LaTeX

La generación de código LaTeX implementa una traducción directa de comandos LangTeX a macros LaTeX predefinidas. El proceso de traducción mapea cada nodo del AST a su equivalente en LaTeX utilizando las macros definidas en `preamble.tex`.

Se decidió que el compilador produzca archivos LaTeX (.tex) autocontenidos que incluyan:

- Un preámbulo completo con todas las dependencias necesarias
- Soporte nativo para fuentes CJK (chino, japonés, coreano)
- Macros personalizadas para cada comando LangTeX
- Configuración automática de paquetes requeridos

Esta decisión fue tomada para ofrecer mayor flexibilidad a los usuarios finales en base al propósito de uso de nuestro compilador: ya sea continuar agregando más contenido en formato LaTeX o generar directamente un PDF. De esta manera, el usuario puede tanto compilar

inmediatamente el documento como integrarlo con materiales LaTeX existentes sin requerir configuraciones adicionales.

Se decidió implementar las macros mediante un symlink a `preamble.tex` en lugar de incluirlas directamente en el prólogo del documento generado. Esta decisión técnica se basó en:

- **Separación de responsabilidades**: Las macros se mantienen en un archivo independiente, siguiendo las convenciones estándar de LaTeX donde el preámbulo y las definiciones de macros se organizan por separado.
- **Legibilidad del documento**: Al mantener las macros en un archivo externo, la parte importante del documento generado permanece limpia y legible, sin ser oscurecida por definiciones técnicas extensas.

Modalidades de Salida

El compilador implementa múltiples modalidades de output basadas en los flags proporcionados:

Sin directorio especificado:

- Output directo a `stdout` para uso en pipelines Unix o visualización inmediata (no imprime el preámbulo)

Con flag `-d` (directorio):

- Crea automáticamente el directorio especificado si no existe
- Genera el archivo dentro del directorio junto con el symlink a `preamble.tex`
- Si no se especifica `-o`, utiliza `main.tex` como nombre por defecto

Con flag `-o` (nombre de archivo):

- Permite especificar el nombre del archivo de salida
- Si no se utiliza `-d` no tiene efecto, y se generará a `stdout`.

Además, se implementó la funcionalidad del parámetro `isInput` como el flag (`-i` o `--input`) para permitir generar solo el contenido sin preámbulo ni epílogo cuando el código LangTeX va a ser insertado dentro de un documento LaTeX existente:

```
if (!isInput) _generatePrologue();
_generateProgram(compilerState->abstractSyntaxtTree);
if (!isInput) _generateEpilogue();
```

Esta decisión permite mayor flexibilidad en el uso del compilador, habilitando tanto la generación de documentos completos como la inserción de fragmentos en documentos existentes.

4. Futuras Extensiones

Como futura extensión, se considera que el presente proyecto podría abarcar más idiomas en la parte de romanización, ofreciendo mayores posibilidades para aprendices de nuevas lenguas. A su vez, se espera poder incluir mayor personalización en el uso de los comandos LaNgTeX, ofreciendo una mayor cantidad de parámetros opcionales (tales como la tipografía, color, alineación, etc), con la finalidad de hacer los comandos más accesibles a personas quienes no hayan tenido experiencias pasadas con Látex. Esto se puede ver en fragmentos de código donde esta contemplado el parámetro style pero sin una implementación.

Por último, se espera implementar herramientas tanto de ayuda visual -como de coloreo, para facilitar la distinción entre comandos, parámetros, y su contenido- como también de ayuda auditiva, para mejorar la pronunciación de las palabras.

5. Conclusiones

Durante el desarrollo del proyecto se pudo poner en práctica el proceso de análisis sintáctico y semántico, con un claro seguimiento mediante la estructura de datos de los árboles. A su vez, se destacó la importancia de un buen diseño de gramática, evitando posibles ambigüedades, tendiendo a la mayor simpleza posible. Por último, se pudo poner en práctica herramientas nuevas, tales como Flex y Bison, y probar el procedimiento de desarrollo de un compilador no desde el lado del usuario, sino del programador.