# RELATÓRIO DO TRABALHO 1
# ANALISADOR LÉXICO

## BCC328 - Construção de compiladores I

**Autores:**
19.2.4004 - Mateus Henrique Máximo Lima de Souza
19.2.4008 - Luiz Fernando Rodrigues Fernandes
24.1

# 1 Introdução

Este trabalho apresenta o desenvolvimento de um analisador léxico utilizando a linguagem de programação Haskell e a ferramenta Alex. O objetivo principal deste relatório é especificar o processo de criação desse componente para a linguagem Lang, garantindo a correta identificação e categorização dos tokens presentes nos programas escritos nessa linguagem.

## 1.1 Análise Léxica

A análise léxica é a primeira fase do processo de compilação, onde o código fonte é transformado em uma sequência de tokens. Cada token representa uma unidade atômica do código, como palavras-chave, identificadores, operadores e literais. A função principal do analisador léxico (lexer) é ler o código fonte, identificar os padrões léxicos e categorizá-los corretamente.

Nesta etapa, o lexer percorre o código fonte caractere por caractere, agrupando-os em tokens conforme as regras léxicas definidas. Essas regras são geralmente especificadas por expressões regulares que descrevem os padrões válidos na linguagem de programação. O lexer deve ser capaz de distinguir entre diferentes tipos de tokens e ignorar espaços em branco e comentários que não são relevantes para a análise sintática (Isidoro 2016).

A correta implementação do analisador léxico é crucial para a construção de um compilador. Erros na análise léxica podem levar a falhas nas etapas subsequentes de análise sintática e semântica. Portanto, é fundamental que o lexer seja preciso e eficiente na categorização dos tokens.

No contexto deste trabalho, utilizamos a ferramenta Alex para gerar o lexer. Alex permite definir padrões léxicos usando expressões regulares e gera automaticamente o código Haskell correspondente. Essa abordagem facilita a implementação do lexer e garante a precisão na identificação dos tokens.

A seguir, apresentamos as ferramentas utilizadas para construção do Lexer para a linguagem Lang.

## 1.2 Ferramentas utilizadas

### 1.2.1 Alex

Alex é uma ferramenta para a geração de analisadores léxicos em Haskell. Ele permite a definição de padrões léxicos usando expressões regulares e gera automaticamente o código Haskell correspondente para a análise desses padrões.

### 1.2.2 Cabal

Cabal é uma ferramenta de gerenciamento de pacotes e construção de projetos Haskell. Ele facilita a especificação das dependências do projeto, configuração de pacotes, construção e teste de projetos Haskell (Developers 2024).

# 2 Desenvolvimento

Nesta seção, apresentamos as principais decisões de projeto tomadas durante o desenvolvimento do analisador léxico para a linguagem Lang. Essas decisões incluem a definição de

expressões regulares para a categorização de tokens, funções auxiliares criadas e a interface para utilização do analisador léxico.

## 2.1 Expressões regulares

A implementação do lexer foi baseada em expressões regulares que definem os padrões para cada tipo de token reconhecido pela linguagem Lang. As expressões regulares foram utilizadas para descrever identificadores, tipos, números inteiros, números de ponto flutuante, literais de caracteres, operadores e palavras-chave da linguagem (Manacero 2024).

Como descrito abaixo:

- A expressão regular para identificar números inteiros `integer` foi definida como `$digit+`, onde `$digit` representa qualquer dígito de 0 a 9. O símbolo `+` indica que a sequência deve conter um ou mais dígitos consecutivos.
- Para números de ponto flutuante (`float`), a expressão regular usada foi `$digit+ "." $digit+ | "." $digit+`, que permite reconhecer tanto números na forma "123.456" quanto na forma ".456".
- As expressões regulares `$lowercase`, `$uppercase`, `$alpha`, e `$underscore` foram usadas para definir as categorias básicas de caracteres:
  - `$lowercase` foi definida como `[a-z]`, representando qualquer letra minúscula do alfabeto.
  - `$uppercase` foi definida como `[A-Z]`, representando qualquer letra maiúscula do alfabeto.
  - `$alpha` combina `$lowercase` e `$uppercase`, sendo definida como `[$lowercase $uppercase]`, o que significa que ela reconhece qualquer letra do alfabeto, seja maiúscula ou minúscula.
  - `$underscore` foi definida como _, representando o caractere sublinhado (underscore).
- Os identificadores (`identifier`) foram definidos como uma letra minúscula seguida de uma combinação de letras, dígitos ou underscores (`$lowercase[$alpha $digit $underscore]`).
- Nome de tipos (`typename`) seguem a convenção de começar com uma letra maiúscula, seguida de letras, dígitos ou underscores (`$uppercase[$alpha $digit $underscore]`).
- Literais de caracteres (`@literalChar`) são caracteres únicos delimitados por aspas simples. Esses foram definidos como um único caractere alfabético ou um caractere especial de escape, como (`\n`), (`\t`), (`\b`), (`\r`), (`\'`), ou (`\\`) através da seguinte expressão regular: `@literalChar = $alpha | \\n | \\t | \\b | \\r | \\ | \\\\ | \'` que, quando combinada com a especificação `"'" @literalChar "'"`, forma um caractere literal válido de acordo com a linguagem Lang.

Essas expressões regulares são automaticamente convertidas em autômatos finitos pela ferramenta Alex, que é usada para gerar o código Haskell responsável pela análise léxica. Cada expressão regular corresponde a um estado do autômato, que percorre o código fonte caractere por caractere, categorizando-o conforme as regras definidas.

É importante ressaltar que a definição completa dos tokens para palavras reservadas, comandos, operadores e outros símbolos específicos podem ser encontrados no arquivo Lexer.x fornecido, e não foram detalhadamente especificados neste relatório.

### 2.2 Funções auxiliares

Para facilitar a leitura de arquivos e a exibição dos tokens gerados como especificado na descrição do trabalho, foram criadas algumas funções auxiliares no Main.hs:

**Função processFile:** Esta função auxilia na leitura do arquivo de entrada, verificando se o arquivo existe e lidando com erros caso o arquivo não seja encontrado. Em seguida, ela passa o conteúdo do arquivo para o lexer, que gera uma lista de tokens.

**Função formatToken:** Esta função formata os tokens gerados pelo Alex, no formato especificado pelo exemplo da Seção 2 do enunciado do trabalho.

**Função printTokens:** Esta função utiliza a função formatToken para imprimir os caracteres formatados e os imprime um por linha.

### 2.3 Interface para utilização do analisador léxico

Para cumprir com os requisitos especificados no trabalho, foi criado um arquivo Main.hs na pasta app. Este arquivo é responsável por receber um arquivo de entrada contendo código na linguagem Lang e, em seguida, imprimir os tokens gerados pelo lexer ou reportar erros caso ocorram durante o processo de análise léxica.

O programa, ao ser executado via **stack run**, oferece ao usuário duas opções:

1. **Utilizar o arquivo `program.lang`:** Esta opção permite que o usuário utilize um arquivo padrão chamado `program.lang`, que está localizado na pasta `app`.
2. **Fornecer um caminho absoluto:** Como alternativa, o usuário pode optar por fornecer o caminho absoluto de um arquivo de sua escolha.

## 3 Execução

Para compilar e executar o analisador léxico da linguagem Lang, utilizamos as ferramentas fornecidas pelo Stack.

O comando **stack run** é utilizado para compilar e executar a aplicação. Este comando é suficiente para verificar se o projeto está atualizado, compilar o código-fonte, se necessário, e em seguida, executar o programa. Durante a execução, o programa oferece ao usuário duas opções, conforme descrito na seção anterior: (1) utilizar um arquivo padrão chamado

`program.lang` localizado na pasta `app` ou (2) fornecer o caminho absoluto de um arquivo de sua escolha.

# 4 Referências

# References


[Dev24] Cabal Developers. *Cabal - The Haskell Package Manager*. Acessado em: 16 de agosto de 2024. 2024. URL: `https://cabal.readthedocs.io/en/stable/`.

[Isi16] John Isidoro. *Análise Lexical*. Accessed: 2024-08-16. 2016. URL: `https://johnidm.gitbooks.io/compiladores-para-humanos/content/part1/lexical-analysis.html`.

[Man24] Aleardo Manacero. *Capítulo 2: Análise Lexical*. Acessado em: 16 de agosto de 2024. 2024. URL: `https://www.dcce.ibilce.unesp.br/~aleardo/cursos/compila/cap02.pdf`.