1. Introdução: Visão Geral do Projeto

Este documento detalha o projeto e a implementação de uma nova linguagem de programação imperativa, desenvolvida para a disciplina DIM0548 - Engenharia de Linguagens da Universidade Federal do Rio Grande do Norte. O principal objetivo do projeto é conceber e implementar uma linguagem de programação, com foco particular na área de Ciência de Dados e Aprendizado de Máquina.

A linguagem foi projetada para ter uma sintaxe clara e expressiva, visando a legibilidade do código. Ela oferecerá suporte a tipagem dinâmica ou estática opcional, provendo flexibilidade e segurança. Um aspecto crucial é o suporte a tipos de dados abstratos, como dataframes e tensores, com sobrecarga de operadores para manipulação eficiente, o que é essencial para o domínio de aplicação.

O sistema de implementação ideal para a linguagem proposta é a interpretação, dada a importância da execução interativa para o desenvolvimento e experimentação em Ciência de Dados e Aprendizado de Máquina. Essa abordagem permite a exploração rápida de dados e o desenvolvimento de modelos sem a necessidade de recompilar a cada modificação, acelerando o processo de prototipagem e ajustes.

A linguagem busca ser um híbrido, herdando características de linguagens como C e GoLang, mantendo um propósito semelhante ao LISP, e se posicionando próxima ao Python na árvore genealógica das linguagens de programação. A inspiração do Python se reflete na sintaxe limpa e expressiva, além do suporte à tipagem estática opcional para controle de desempenho e segurança. O projeto visa equilibrar facilidade de uso e expressividade com a eficiência e robustez computacional de linguagens compiladas, tornando-a atrativa tanto para cientistas de dados quanto para desenvolvedores que necessitam de alto desempenho.

O projeto aborda as fases principais de um compilador, incluindo a análise léxica e a análise sintática, com a construção de um analisador sintático bottom-up (LALR). A linguagem também define suas vinculações e sistema de tipos, com foco em tipagem estática e forte.

Palavras Reservadas da Linguagem G2DL

Palavras reservadas são identificadores especiais que **não podem ser usados como nomes de variáveis, funções ou objetos**, pois possuem um **significado fixo** na linguagem. Elas fazem parte da **gramática principal** e são reconhecidas diretamente pelo analisador léxico.

Abaixo está a lista completa das palavras reservadas da linguagem G2DL, junto com uma descrição do papel de cada uma.

Controle de Fluxo

Palavra	Função
`if`	Inicia uma estrutura condicional. Executa um bloco se a condição for verdadeira.
`else`	Bloco alternativo executado se a condição do `if` for falsa.
`while`	Laço que executa um bloco enquanto a condição for verdadeira.
`for`	Estrutura de repetição com inicialização, condição e incremento definidos.
`break`	Interrompe imediatamente a execução do laço atual (`for` ou `while`).

Funções e Retornos

Palavra	Função
`function`	Define uma função com nome, parâmetros e corpo.
`return`	Encerra a função e opcionalmente retorna um valor.

Sintaxe

Documentação da AST

Estrutura da AST (`ast.c`)

Este módulo implementa as funções de construção e gerenciamento da Árvore de Sintaxe Abstrata (AST) da linguagem. Ele é responsável por representar a estrutura lógica do código-fonte após a análise sintática, permitindo que fases posteriores como análise semântica e interpretação/execução sejam realizadas.

Estrutura Geral

- **Arquivo:** `ast.c`
- **Cabealhos:** `#include "ast.h"`
- **Uso de memória:** Todas as alocações so feitas via `malloc`, `realloc`, `strdup`, e liberadas com `free`.
- **Origem da linha:** Cada n da AST marcado com a varivel global `yylineno` (linha atual do lexer/parser).

Criação de Nós da AST

Todos os nós são derivados de `AstNode` e criados com uma função base: static AstNode* create_base_node(AstNodeType type, size_t size); Ela aloca memoria, atribui o tipo do nó e o número da linha.

Tipos de Nós Literais

Função	Tipo de Nó	Descrição
`create_int_literal_node(int)`	`NODE_TYPE_INT_LITERAL`	Representa um número inteiro.
`create_float_literal_node(float)`	`NODE_TYPE_FLOAT_LITERAL`	Representa um número de ponto flutuante.

Execução

1. Instruções de Uso do Compilador

Comandos Make Disponíveis

Você pode usar os seguintes comandos com make:

Comando	O que faz
make ou make all	Compila todo o projeto e gera bin/g2dl_interpreter
make clean	Remove todos os arquivos gerados, incluindo build/, bin/ e temporários
make debug	Não implementado (mas pode ser usado para recompilar com -g)
make help	Exibe instruções de uso dos comandos do Makefile

Para gerar e utilizar o compilador G2DL, siga os passos abaixo: Passo 1: Gerar o Compilador Para compilar o código-fonte da linguagem e gerar o executável do compilador, navegue até o diretório raiz do projeto no seu terminal e execute o comando make: Bash $ make

Esse comando executa os seguintes passos automaticamente:

• Gera o analisador léxico com Flex

• Gera o analisador sintático com Bison

• Compila todos os arquivos .c

• Organiza os .o em build/

• Cria o executável final em bin/g2dl_interpreter

Passo 2: Executar o Compilador Após a compilação, o executável do compilador será gerado. Você pode utilizá-lo para processar um arquivo de código-fonte da linguagem G2DL. Por exemplo, para compilar o arquivo input.txt, utilize o seguinte comando: Bash $ ./compiler_g2dl input.txt

Certifique-se de substituir input.txt pelo nome do arquivo de código-fonte que deseja compilar. Passo Extra: Limpar o Compilador Para remover os arquivos gerados durante o processo de