Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada
DIM0548 - Engenharia de Linguagens - T01

Problema: Projetar e implementar uma linguagem de programação

Docente:
Prof. Dr. Umberto Souza de Costa

Discentes:
Carlos Eduardo Miranda da Silva
Danilo de Souza Braga Aciole
Ignácio Saglio Rossini
Rary Emanuel Gonçalves Coringa

Natal-RN, 2025

Documentação do Analisador Sintático

1. *Estrutura dos Programas da Linguagem Proposta*

A linguagem **DataLang** foi projetada com sintaxe baseada na língua portuguesa, utilizando palavras-chave como `funcao`, `se`, `para`, e `enquanto`. A estrutura de blocos é definida por palavras-chave de fechamento, como `fimFuncao` e `fimSe`.

1.1. Código Mínimo Compilável

A unidade fundamental da DataLang é a `funcao`. Um programa é composto por uma lista de definições de função. Embora um arquivo vazio possa ser tecnicamente aceito pela gramática (uma lista de 0 funções), um programa mínimo "útil" que pode ser executado deve conter uma função `Main`.

1.2. Exemplos Gerais

A DataLang suporta estruturas de controle, tipos complexos (como `Lista<Inteiro>`), programação procedural e expressões complexas, incluindo chamadas de método (`.tamanho()`) e acesso a índices (`[i]`).

```
funcao Lista<Inteiro> merge(Lista<Inteiro> esquerda,
Lista<Inteiro> direita)
    Lista<Inteiro> resultado <- [];
    Inteiro indice_esq <- 0;
    Inteiro indice_dir <- 0;

    enquanto (indice_esq < esquerda.tamanho() && indice_dir <
direita.tamanho())
        se (esquerda[indice_esq] < direita[indice_dir])
            resultado.adicionar(esquerda[indice_esq]);
            indice_esq += 1;
        senao
            resultado.adicionar(direita[indice_dir]);
            indice_dir += 1;
        fimSe
    fimEnquanto

    enquanto (indice_esq < esquerda.tamanho())
        resultado.adicionar(esquerda[indice_esq]);
        indice_esq += 1;
    fimEnquanto

    enquanto (indice_dir < direita.tamanho())
```

```
        resultado.adicionar(direita[indice_dir]);
        indice_dir += 1;
    fimEnquanto

    retorne resultado;
fimFuncao

funcao Lista<Inteiro> mergeSort(Lista<Inteiro> lista)
    se (lista.tamanho() <= 1)
        retorne lista;
    fimSe

    Inteiro meio <- lista.tamanho() / 2;
    Lista<Inteiro> sublista_esquerda <- [];
    Lista<Inteiro> sublista_direita <- [];

    para (Inteiro i <- 0; i < meio; i += 1)
        sublista_esquerda.adicionar(lista[i]);
    fimPara

    para (Inteiro i <- meio; i < lista.tamanho(); i += 1)
        sublista_direita.adicionar(lista[i]);
    fimPara

    sublista_esquerda <- mergeSort(sublista_esquerda);
    sublista_direita <- mergeSort(sublista_direita);

    retorne merge(sublista_esquerda, sublista_direita);
fimFuncao

funcao Inteiro Main()
    Lista<Inteiro> numeros_desordenados <- [59, 7, 1, 10, 2,
17, 8];
    escreva("Lista Original: " +
paraTexto(numeros_desordenados));

    Lista<Inteiro> numeros_ordenados <-
mergeSort(numeros_desordenados);
    escreva("Lista Ordenada: " +
paraTexto(numeros_ordenados));
fimFuncao
```

2. Estrutura dos analisadores

O processo de compilação é dividido em duas fases principais: a análise léxica (Flex) e a análise sintática (Bison).

2.1. Analisador Léxico

O analisador léxico (`scanner.l`) é responsável por ler o código-fonte da DataLang e agrupá-lo em uma sequência de *tokens*.

2.2. Analisador Sintático

O analisador sintático (`parser.y`) recebe os tokens do léxico e verifica se eles obedecem à gramática formal da DataLang.

3. Uso do Analisador Sintático

Este guia descreve os passos necessários para gerar, compilar e executar o analisador léxico e sintático da linguagem DataLang utilizando Flex, Bison e GCC.

Requisitos

Certifique-se de ter os seguintes pacotes instalados em seu sistema: Flex, Bison e o GCC

Passo a Passo (make)

Este comando executa automaticamente todos os passos necessários para construir o compilador, incluindo a criação da pasta de build, geração do scanner e parser, e compilação final.

```
make build
```

Passo a Passo (manual)

Caso prefira não usar o justfile, você pode executar manualmente os comandos abaixo. Todos os passos funcionam da mesma forma, apenas requerem execução individual de cada comando.

1. **Criar diretório**

   Cria um diretório separado para organizar todos os arquivos intermediários e o executável final, mantendo o projeto limpo.

   ```
   $ mkdir -p build
   ```

2. **Gere o código C do scanner com Flex**

O Flex processa o arquivo scanner.l (definição de tokens e regras léxicas) e gera o código C do analisador léxico, responsável por identificar tokens no código fonte.

```
flex -o build/scanner.yy.c scanner.l
```

3. Gere o código C do parser com Bison

O Bison processa o arquivo parser.y (gramática e regras sintáticas) e gera o código C do analisador sintático, que verifica se a sequência de tokens segue a estrutura gramatical da linguagem. As flags -d, -v e -g geram arquivos auxiliares (cabeçalhos, relatórios e grafos).

```
$ bison -o build/parser.tab.c -d -v -g parser.y
```

4. Gere o compilador com GCC

Compila os arquivos C gerados pelo Flex e Bison em um único executável, criando o compilador final da linguagem DataLang.

```
gcc build/scanner.yy.c build/parser.tab.c -o
build/compiler
```