Compilador para Kotlin-Parte 1

0. Identificação

Grupo: T02_G05

Nomes: André Filipe Carvalho Guedes Borges Adrêgo
Bruno Alexandre Magalhães Costa

1. Introdução

Este projeto implementa um compilador simples para um subconjunto da linguagem Kotlin, focando em variáveis, tipos básicos (inteiro, ponto flutuante, booleano, string), operações aritméticas e lógicas, estruturas de controle (if, while), e entrada/saída (readln, print). Este compilador usa **Alex** para análise léxica e **Happy** para análise sintática, escrito em Haskell.

2. Objetivos do Projeto

- Desenvolver uma análise léxica para identificar tokens.
- Implementar um parser que reconheça a estrutura da linguagem e gere uma árvore sintática abstrata (AST).
- Definir nós da AST que representem corretamente as expressões e comandos do subconjunto de Kotlin.
- Implementar um tratamento básico de erros de parsing.

3. Estrutura do Projeto

- **Lexer.x**: Define os tokens e as expressões regulares.
- **Parser.y**: Define a gramática da linguagem, com precedências e AST.
- **Main.hs**: Função principal para inicializar a análise.
- **README.pdf**: Documento de descrição detalhada do projeto.

4. Funcionalidades da Linguagem

A linguagem aceita:

- **Declaração de variáveis e constantes**: val para constantes imutáveis e var para variáveis mutáveis.
- **Tipos suportados**: Int, Float, Boolean, String.
- **Operadores**:
    - **Aritméticos**: +, -, *, /, %.
    - **Lógicos**: &&, ||, !.
    - **Comparativos**: ==, !=, <, <=, >, >=.
- **Controle de Fluxo**: if, else, while.
- **Entrada e Saída**: readln(), print().

5. Instruções para Compilação e Execução

Para compilar e executar o projeto:

1. Gere o analisador léxico:
   alex Lexer.x
2. Gere o parser:
   happy Parser.y
3. Compile o projeto com o GHC:
   ghc Main.hs
4. Execute o compilador:
   ./Main < exemplo > resultado

6. Análise Léxica (Lexer)

O arquivo Lexer.x define os tokens utilizando expressões regulares, identificando palavras-chave, operadores, delimitadores e literais da linguagem. Abaixo, estão todos os tokens definidos no analisador léxico:

6.1 Palavras-chave

As palavras-chave são reservadas e não podem ser usadas como identificadores:

- **if**: Palavra-chave para estruturas condicionais.
- **else**: Palavra-chave para cláusulas else.
- **while**: Palavra-chave para laços while.
- **fun**: Define uma função.
- **main**: Nome da função principal.
- **val**: Declara uma constante imutável.
- **var**: Declara uma variável mutável.
- **Int, Float, Boolean, String**: Tipos de dados suportados.

6.2 Identificadores e Literais

- **Identificadores**: Sequências de letras e dígitos (começando por uma letra ou _), representados pelo token ID.
- **Números Inteiros**: Sequências de dígitos (ex: 123) com o token NUM.
- **Números Reais**: Números com ponto decimal (ex: 12.34), representados pelo token REAL.
- **Strings**: Sequências de caracteres entre aspas duplas, como "texto", mapeados para o token STR.
- **Booleanos**: true e false, mapeados para os tokens TRUE e FALSE.

6.3 Operadores

➔ **Operadores Aritméticos**:
- ◆ +: Soma (PLUS)
- ◆ -: Subtração (MINUS)
- ◆ *: Multiplicação (MULT)
- ◆ /: Divisão (DIV)
- ◆ %: Módulo (MOD)

➔ **Operadores de Comparação**:
- ◆ ==: Igualdade (EQUAL)
- ◆ !=: Diferente (NEQUAL)
- ◆ <: Menor que (L)
- ◆ <=: Menor ou igual (LEQ)
- ◆ >: Maior que (G)
- ◆ >=: Maior ou igual (GEQ)

➔ **Operadores Lógicos**:
- ◆ &&: E lógico (AND)
- ◆ ||: Ou lógico (OR)
- ◆ !: Negação (NOT)

➔ **Operadores de Incremento/Decremento**:
- ◆ ++: Incremento (ICR)
- ◆ --: Decremento (DCR)

➔ **Operadores de Atribuição**:
- ◆ =: Atribuição (ATRIB)
- ◆ +=: Atribuição com soma (ATRIB_PLUS)
- ◆ -=: Atribuição com subtração (ATRIB_MINUS)
- ◆ *=: Atribuição com multiplicação (ATRIB_MULT)
- ◆ /=: Atribuição com divisão (ATRIB_DIV)
- ◆ %=: Atribuição com módulo (ATRIB_MOD)

6.4 Delimitadores

- (e): Parênteses para expressões e chamadas de função, mapeados para LPAREN e RPAREN.
- { e }: Delimitadores de blocos de código, mapeados para LBRACE e RBRACE.
- **:** : Delimitador de anotação de tipo, mapeado para COLON.
- **;** : Delimitador de final de comando, mapeado para SEMICOLON.

6.5 Funções de Entrada e Saída

- **Entrada**: readln, usado para leitura de dados, mapeado para READLN.
- **Saída**: print, usado para exibir dados, mapeado para PRINT.

7. Análise Sintática e AST

O arquivo Parser.y define a gramática da linguagem e como cada expressão e comando se traduz em um nó da AST (Árvore Sintática Abstrata). Abaixo estão descritos os principais tipos de nós da AST e o que representam:

7.1 Nós para Declaração de Variáveis e Constantes

- **VarDecl**: Declara uma variável mutável com inicialização.
- **VarDeclTyped**: Declara uma variável mutável com um tipo explícito e inicialização.
- **ValDecl**: Declara uma constante com inicialização.
- **ValDeclTyped**: Declara uma constante com um tipo explícito e inicialização.

7.2 Nós para Estruturas de Controle

- **IfNode**: Representa uma estrutura if com uma condição e bloco de comandos.
- **IfElseNode**: Representa uma estrutura if com else, incluindo condição, bloco if e bloco else.
- **WhileNode**: Representa um laço while com condição e corpo.

7.3 Nós para Operações Aritméticas e Lógicas

- **Aritméticos**:
    - **AddNode**: Representa uma soma.
    - **SubNode**: Representa uma subtração.
    - **MultNode**: Representa uma multiplicação.
    - **DivNode**: Representa uma divisão.
    - **ModNode**: Representa uma operação de módulo.

- **Lógicos e Comparativos**:
    - **AndNode** e **OrNode**: Representam operações lógicas && e ||.
    - **GtNode**, **GeNode**, **LtNode**, **LeNode**: Representam comparações >, >=, <, <=.
    - **EqNode** e **NeNode**: Representam igualdade (==) e diferença (!=).
    - **NotNode**: Representa uma negação lógica.
- **Incremento e Decremento**:
    - **IncrNode**: Representa o operador de incremento ++.
    - **DecrNode**: Representa o operador de decremento --.

7.4 Nós para Atribuição

Cada operador de atribuição possui um nó específico:

- **AssignNode**: Atribuição direta =.
- **AddAssignNode**: Atribuição com soma +=.
- **SubAssignNode**: Atribuição com subtração -=.
- **MultAssignNode**: Atribuição com multiplicação *=.
- **DivAssignNode**: Atribuição com divisão /=.
- **ModAssignNode**: Atribuição com módulo %=.

7.5 Nós para Entrada e Saída

- **PrintNode**: Representa a função print para exibir valores.
- **ReadlnNode**: Representa a função readln para leitura de valores.

7.6 Literais e Tipos Básicos

- **NumNode**: Representa números inteiros.
- **RealNode**: Representa números reais (ponto flutuante).
- **StringNode**: Representa strings.
- **BoolNode**: Representa valores booleanos (true ou false).
- **IdNode**: Representa identificadores (variáveis e constantes).

8. Tratamento de Erros

Quando o parser encontra um erro, ele interrompe a análise e imprime os tokens restantes que não foram processados. Essa informação auxilia na localização da posição do erro no código.

9. Conclusão

Este projeto de compilador para um subconjunto da linguagem Kotlin proporcionou uma visão prática dos processos fundamentais envolvidos no desenvolvimento de compiladores, incluindo a análise léxica, a análise sintática e a geração de uma árvore sintática abstrata (AST). O projeto alcançou com sucesso os objetivos de reconhecer e processar estruturas típicas de uma linguagem de programação.