Ciência da Computação
Instituto Federal do Ceará
Campus Maracanaú

Professor: Hericson Araújo
hericson.araujo@ifce.edu.br
Disciplina: Compiladores

Trabalho Final - Compiladores 2025.1

1 Descrição do Projeto

Trabalhando em equipes de **Três Pessoas** desenvolva um compilador para a linguagem didática **TurtleScript**, com foco em análise léxica, sintática, semântica e geração de código Python com **Turtle Graphics**.

2 Descrição da Biblioteca Turtle em Python

A **Turtle Graphics** é uma biblioteca padrão do Python que permite a criação de desenhos vetoriais de forma simples e intuitiva. Ela é amplamente utilizada no ensino de programação por ser muito visual. A ideia principal é controlar um cursor, chamado de "tartaruga" *turtle*), que se move em uma tela cartesiana (*screen*).
Ao receber comandos como "avançar 100 pixels"ou "girar 90 graus", a tartaruga se movimenta, deixando um rastro por onde passa (se sua "caneta" estiver abaixada). No contexto deste projeto, a linguagem personalizada **TurtleScript** é compilada para um conjunto de comandos que a biblioteca Turtle pode interpretar e executar, traduzindo a lógica abstrata da nova linguagem em um resultado gráfico concreto.

3 Objetivo Geral

Desenvolver as etapas (léxica, sintática e semântica) de um compilador para uma linguagem personalizada chamada: **TurtleScript**, utilizando um parser recursivo descendente para linguagens LL(1). O compilador deverá incluir suporte à análise semântica com verificação de tipos, e gerar como saída código Python funcional que utiliza a biblioteca Turtle Graphics para a simulação visual do programa de entrada. Ao final, a linguagem personalizada **TurtleScript** deverá ser traduzida para a linguagem Python com a biblioteca: **Turtle Graphics**.

3.1 Visão Geral e Conceitos

Esta seção detalha a estrutura e os componentes da **TurtleScript**, a linguagem a ser implementada. A **TurtleScript** controla um cursor (a "tartaruga") numa tela bidimensional. O objetivo do compilador é traduzir um programa escrito em TurtleScript para um script Python funcional que desenha o resultado esperado.

4 Linguagem Personalizada

A linguagem proposta permite comandos simples para movimentação gráfica, como:

- `avancar <número>;`
- `girar_direita <número>;`
- `var inteiro:  x, y;`
- `x = 100;`

O bloco principal do programa inicia com `inicio` e termina com `fim`.

4.1 Tipos de Dados

Esta seção detalha os tipos de dados suportados pela linguagem `TurtleScript`

`inteiro` Utilizado para valores numéricos inteiros. É aplicado em comandos de movimento para especificar distâncias, em comandos de rotação para ângulos e no comando de repetição.

`texto` Utilizado para valores de string. É essencial para definir cores por nome e em outros comandos que manipulam texto.

`real` Tipo de dado para representar números de ponto flutuante. Permitiria maior precisão em distâncias e ângulos (ex: `avancar 10.5;` ou `girar_direita 22.5;`).

`logico` Tipo de dado booleano que pode assumir os valores `verdadeiro` ou `falso`. Seria a base para novas estruturas de controle condicionais (ex: `se...fim_se`).

4.2 Estrutura do Programa

Todo programa em TurtleScript deve ser contido dentro de um bloco principal delimitado.

Comando	Descrição
`inicio` ... `fim`	Delimita o escopo principal do programa. Todo o código executável e as declarações de variáveis devem estar entre estas duas palavras-chave.

Exemplo de Uso

```
1 inicio
2 // Seu codigo vai aqui
3 fim
```

4.3 Declaração e Atribuição de Variáveis

A linguagem suporta variáveis com tipagem estática, que devem ser declaradas no início do escopo.

Sintaxe	Descrição
`var` tipo: id1, ...;	Declara uma ou mais variáveis de um determinado tipo. Os tipos suportados são `inteiro`, `texto`, `real` e `logico`.
`id = <expressao>;`	Atribui um valor a uma variável já declarada. A expressão pode ser um literal ou outra variável

4.4 Exemplo de Uso

```
inicio
    // --- Declaracao de Variaveis ---
    // A declaracao de todas as variaveis ocorre no inicio do escopo.
    var inteiro: passo, repeticoes;
    var texto: cor_fundo, cor_linha, titulo;
    var real: angulo_preciso;
    var logico: desenhar_forma;

    // --- Atribuicoes de Valores ---
    // Atribuicao a variaveis do tipo 'inteiro'.
    passo = 150;
    repeticoes = 5;

    // Atribuicao a variaveis do tipo 'texto' com string.
    titulo = "Desenho de uma Estrela";
    cor_fundo = "black";
    cor_linha = "blue";

    // Atribuicao a uma variavel do tipo 'real' para maior precisao.
    angulo_preciso = 144.0;

    // Atribuicao a uma variavel do tipo 'logico'.
    desenhar_forma = verdadeiro;

fim
```

Listing 1: Exemplo de uso com todos os tipos de dados.

4.5 Sintaxe da Linguagem Personalizada

Para aumentar o poder de expressão da linguagem `TurtleScript`, foram propostas novas estruturas de controle que permitem a execução de código de forma condicional e a criação de laços dinâmicos.

Estrutura Condicional: `se`

A estrutura `se/senao` permite que o programa tome decisões e execute blocos de código diferentes com base no resultado de uma expressão lógica. Isso é fundamental para a criação de algoritmos não triviais.

Sintaxe	Descrição

`se <expr_logica> entao`
  ...
`fim_se`
	Executa um bloco de comandos se a expressão lógica for `verdadeiro`.

`se <expr_logica> entao`
  ...
`senao`
  ...
`fim_se`
	Executa o primeiro bloco de comandos se a expressão for `verdadeiro`; caso contrário, executa o bloco após o `senao`.

Exemplo de Uso

```
inicio
    var inteiro: contador = 0;

    repita 10 vezes
        // Verifica se o contador e par ou impar
        se (contador % 2) == 0 entao
            definir_cor "cyan";
        senao
            definir_cor "yellow";
        fim_se;

        avancar 25;
        contador = contador + 1;
    fim_repita;
fim
```

Listing 2: Uso do condicional para alternar cores.

Estrutura de Repetição Condicional: `enquanto`

Diferente do laço `repita`, que executa um número fixo de vezes, a estrutura `enquanto` executa um bloco de comandos continuamente enquanto uma condição lógica permanecer `verdadeiro`. Isso permite criar laços cujo número de iterações não é conhecido previamente.

Sintaxe	Descrição

`enquanto <expr_logica>`
`faca`
  ...
`fim_enquanto`
	Enquanto a expressão lógica for avaliada como `verdadeiro`, o bloco de comandos interno é executado repetidamente.

Exemplo de Uso

```
inicio
    var inteiro: lado = 10;
    cor_de_fundo "black";
    definir_cor "white";

    // O laco continua apenas enquanto o lado for menor que 200
    enquanto lado < 200 faca
```

```
8           avancar lado;
9           girar_direita 91;
10
11          // Incrementa a variavel de controle do laco
12          lado = lado + 2;
13      fim_enquanto;
14  fim
```

Listing 3: Desenhando uma espiral que cresce até um limite.

Comandos de Movimento

Sintaxe	Descrição
`avancar <expr>;`	Move a tartaruga para frente pela distância especificada.
`recuar <expr>;`	Move a tartaruga para trás pela distância especificada.
`girar_direita <expr>;`	Gira a tartaruga para a direita pelo ângulo especificado.
`girar_esquerda <expr>;`	Gira a tartaruga para a esquerda pelo ângulo especificado.
`ir_para <expr> <expr>;`	Move a tartaruga para as coordenadas absolutas (x, y).

Comandos de Controle da Caneta

Sintaxe	Descrição
`levantar_caneta;`	Levanta a caneta, movendo sem desenhar.
`abaixar_caneta;`	Abaixa a caneta, voltando a desenhar.
`definir_cor <expr>;`	Define a cor da linha (ex: "red").
`definir_espessura <expr>;`	Define a espessura da linha em pixels.

Comandos de Controle de Tela

Sintaxe	Descrição
`cor_de_fundo <expr>;`	Define a cor de fundo da tela.
`limpar_tela;`	Apaga todos os desenhos da tela.

Estrutura de Controle: Repetição

Sintaxe	Descrição
`repita <num> vezes ... fim_repita;`	Executa o bloco de comandos um número fixo de vezes.

Exemplo de Uso

```
1 // Desenha um pentagono
2 repita 5 vezes
3     avancar 100;
4     girar_direita 72;
5 fim_repita;
```

4.6 Catálogo de Comandos

Categoria	Comando TurtleScript	Exemplo de Uso	Código Python Gerado

Movimento	`avancar <expr>`
	`recuar <expr>`
	`girar_direita <expr>`
	`girar_esquerda <expr>`
	`ir_para <expr> <expr>`
		`avancar 100;`
		`recuar x;`
		`girar_direita 90;`
		`girar_esquerda 45;`
		`ir_para 0 50;`
			`turtle.forward(100)`
			`turtle.backward(x)`
			`turtle.right(90)`
			`turtle.left(45)`
			`turtle.goto(0, 50)`

Controle da Caneta	`levantar_caneta`
	`abaixar_caneta`
	`definir_cor <expr>`
	`definir_espessura <expr>`
		`levantar_caneta;`
		`abaixar_caneta;`
		`definir_cor "blue";`
		`definir_espessura 3;`
			`turtle.penup()`
			`turtle.pendown()`
			`turtle.pencolor("blue")`
			`turtle.pensize(3)`

Controle de Tela	`limpar_tela` `cor_de_fundo <expr>`	`limpar_tela;` `cor_de_fundo "black";`	`turtle.clear()` `turtle.bgcolor("black")`

Estruturas de Controle	`repita <num> vezes`
	`se <expr> entao`
	
	`se ... senao`
	
	`enquanto <expr> faca`
		`repita 4 vezes`
		`se x > 10 entao`
		
		`se cor=="azul" senao`
		
		`enquanto i < 10 faca`
			`for _ in range(4):`
			`if x > 10:`
			`if cor=="azul":`
			`...`
			`else:`
			`while i < 10:`

4.7 Exemplo 1: Desenhando um Quadrado Simples

Este é o caso mais básico, usando apenas comandos de movimento diretos.

TurtleScript (`entrada1.txt`)

```
inicio
    // Desenha as quatro arestas do quadrado
    avancar 150;
    girar_direita 90;

    avancar 150;
    girar_direita 90;

    avancar 150;
    girar_direita 90;

    avancar 150;
    girar_direita 90;
fim
```

Listing 4: Código para desenhar um quadrado.

Python Gerado (`saida1.py`)

```
import turtle

# --- Configuracao Padrao ---
screen = turtle.Screen()
```

```
t = turtle.Turtle()
screen.title("Resultado - Exemplo 1")

# --- Codigo Gerado pelo Compilador ---
t.forward(150)
t.right(90)
t.forward(150)
t.right(90)
t.forward(150)
t.right(90)
t.forward(150)
t.right(90)

# --- Finalizacao ---
turtle.done()
```

Listing 5: Código Python gerado pelo compilador.

4.8 Exemplo 2: Desenhando uma Estrela com Variáveis

Este exemplo introduz a declaração e o uso de variáveis.

TurtleScript (`entrada2.txt`)

```
inicio
    var inteiro: tamanho_lado;
    tamanho_lado = 200;

    // Desenha uma estrela de 5 pontas
    avancar tamanho_lado;
    girar_direita 144;

    avancar tamanho_lado;
    girar_direita 144;

    avancar tamanho_lado;
    girar_direita 144;

    avancar tamanho_lado;
    girar_direita 144;

    avancar tamanho_lado;
    girar_direita 144;
fim
```

Listing 6: Código para desenhar uma estrela de 5 pontas.

Python Gerado (`saida2.py`)

```
import turtle

screen = turtle.Screen()
t = turtle.Turtle()
screen.title("Resultado - Exemplo 2")

# Declaracao de variaveis
tamanho_lado = 0

# Atribuicao de variaveis
tamanho_lado = 200

# Comandos de desenho
t.forward(tamanho_lado)
t.right(144)
t.forward(tamanho_lado)
t.right(144)
t.forward(tamanho_lado)
t.right(144)
t.forward(tamanho_lado)
t.right(144)
t.forward(tamanho_lado)
t.right(144)
```

```
24
25 turtle.done()
```

Listing 7: Código Python com uso de variáveis.

4.9 Exemplo 3: Espiral com Laços e Cores

Este exemplo avançado utiliza laços de repetição, variáveis e controle da caneta.

TurtleScript (`entrada3.txt`)

```
inicio
    var inteiro: lado;
    var texto: cor;

    lado = 5;
    cor_de_fundo "black";
    definir_espessura 2;

    repita 50 vezes
        // Muda a cor da linha a cada iteracao
        definir_cor "cyan";

        // Desenha e aumenta o lado
        avancar lado;
        girar_direita 90;
        lado = lado + 5;
    fim_repita;
fim
```

Listing 8: Código para desenhar uma espiral colorida.

Python Gerado (`saida3.py`)

```
import turtle

screen = turtle.Screen()
t = turtle.Turtle()
screen.title("Resultado - Exemplo 3")
t.speed(0) # Velocidade maxima

# Declaracao de variaveis
lado = 0
cor = ""

# Atribuicoes iniciais
lado = 5
screen.bgcolor("black")
t.pensize(2)

# Laco de repeticao
for _ in range(50):
    t.pencolor("cyan")

    # Comandos de desenho e atualizacao
    t.forward(lado)
    t.right(90)
    lado = lado + 5

```

```
26 turtle.done()
```

Listing 9: Código Python com laço 'for'.

5 Análise Léxica e Sintática

(a) **Tradução para Python** A linguagem personalizada **TurtleScript** deverá ser traduzida para código Python compatível com a biblioteca **TurtleGraphics.**

(b) **Implementação do Tradutor:** O tradutor será implementado pela equipe. Ele será dividido em **analisador léxico**, **analisador sintático**, **analisador semântico** e **gerador de código**.

(c) Caso haja algum erro léxico,sintático ou semântico, deverá ser informado.

(d) Crie um arquivo **.sh** que automatize o processo.

(e) Gere um arquivo em **Python** a partir do código programado na **linguagem personalizada**.

6 Análise Semântica

Após a construção da Árvore Sintática Abstrata (AST), o analisador semântico percorre a árvore para verificar a coerência e o significado do programa. Para a TurtleScript, as seguintes verificações são essenciais e devem ser implementadas através de uma Tabela de Símbolos.

6.1 Verificação de Declaração de Variáveis

A Tabela de Símbolos armazenará todas as variáveis declaradas, juntamente com seus tipos.

- **Uso antes da declaração:** Para cada variável encontrada em uma expressão ou atribuição, o analisador deve verificar se ela existe na Tabela de Símbolos. Caso contrário, um erro de "variável não declarada"deve ser reportado.
- **Redeclaração de variável:** Ao processar uma declaração, o analisador deve verificar se a variável já existe na Tabela de Símbolos. Se existir, um erro de "variável já declarada"deve ser emitido.

6.2 Verificação de Tipos (Tipagem Estática)

- **Atribuição:** Em um comando de atribuição como `x = y;`, o tipo da variável `y` (ou do valor literal) deve ser compatível com o tipo da variável `x`, conforme registrado na Tabela de Símbolos. Um erro de "tipos incompatíveis"deve ser gerado se, por exemplo, tentar-se atribuir um `texto` a uma variável do tipo `inteiro` ou `real`.
- **Argumentos de Comandos:** Os tipos das expressões passadas como argumentos para os comandos devem ser validados. Por exemplo, o comando `avancar` espera um argumento do tipo `inteiro`. Se uma variável do tipo `texto` for fornecida, um erro semântico deve ser acusado.

7 Componentes Obrigatórios do Projeto

1.) Tokenizador: Implementação manual para reconhecer todos os elementos da linguagem personalizad **TurtleScript**.

2.) Parser Recursivo Descendente LL(1) Implementação manual com construção de uma Árvore Sintática Abstrata (AST).

3.) Análise Semântica: O analisador deve implementar:

- Verificação de declaração prévia de variáveis (escopo).
- Verificação de tipos (tipagem estática: inteiro/texto).
- Proibição de comandos com tipos de argumentos inválidos.

4.) Geração de Código: Geração de código Python com a biblioteca Turtle a partir da AST, criando arquivos `.py` executáveis.

5.) Casos de Teste: Criar e testar pelo menos 3 programas diferentes em **TurtleScript**.

6.) Novos Comandos: Crie pelo menos dois comandos novos

7.1 Análise de Comandos Específicos

- **Comando `ir_para`:** Deve receber exatamente dois argumentos, ambos do tipo `inteiro`.
- **Comando `definir_cor`:** Deve receber um argumento do tipo `texto`.
- **Comando `repita`:** O número de repetições deve ser um literal do tipo `inteiro`, não uma variável.

7.2 Entregáveis

- **Código-fonte** modularizado: `tokenizer.py`, `parser.py`, `semantico.py`, `gerador.py`.
- **Arquivos de teste** de entrada: `entrada1.txt`, `entrada2.txt`, etc.
- **Códigos Python gerados**: `saida1.py`, `saida2.py`, etc.
- **Relatório final** em formato PDF.
- **Slide da Apresentação**
- **(Opcional)** Script de automação da execução (ex: `execucao.sh`).
- **(Opcional - Nota Bônus)**: Substitua a implementação do *parser* recursivo descendente por um parser LL(1) dirigido por tabela.

7.3 Estrutura do Relatório Final

O relatório deverá conter, no mínimo:

- Descrição detalhada da linguagem **TurtleScript**.
- O passo a passo da **implementação**
- A **Gramática** e as regras de *parsing* implementadas.
- A estratégia utilizada na **análise semântica**.
- Detalhes sobre o processo de **geração de código**.
- **Casos de teste** com resultados, prints dos desenhos gerados e comentários.
- Discussão sobre as **dificuldades enfrentadas** e as soluções encontradas.

7.4 Observações Finais

- **Trabalhos feitos majoritariamente por IA serão anulados**
- **Plágio resultará em nota zero.** Cada aluno deve ser capaz de explicar qualquer parte do projeto.
- **Verifique se a gramática original da linguagem é LL(1), caso não seja, realize as modificações necessárias.**

8 Cronograma

O projeto possui as seguintes datas relevantes previstas:

- **18/06/2025 a 07/07/2025**: Finalização dos Projetos
- **08/07/2025**: Entrega do projeto final pelo *Classroom*
  - Código
  - Relatório
  - Apresentação
- **09/07/2025 e 11/07/2025**: Apresentação dos trabalhos
- **16/07/2025**: Prova Final

Data de entrega final: 08/07/2025