UNIVERSIDADE ESTADUAL DO MARANHÃO **Teoria da Computação e Compiladores** **3ª AVALIAÇÃO - PROJETO**	
Professor: EDILSON LIMA	
Matrícula / Nome Aluno:	

PROJETO

A seguir vai as instruções completa, clara e didática para o **projeto de compiladores** em Python, usando **Google Colab**, totalmente alinhada ao conteúdo das aulas anexadas (análise léxica, sintática, semântica, tabela de símbolos e geração de código).

Projeto Final: Construção de um Compilador em Python (Portugol → C)

Este projeto desafia os alunos a aplicar **todas as etapas estudadas** — análise léxica, sintática, semântica, tabela de símbolos e geração de código intermediário — para construir um **mini-compilador** que traduza um subconjunto de **Portugol** para **linguagem C**.

O desenvolvimento será feito em **Google Colab**, permitindo que todos executem o compilador sem instalar nada localmente.

Objetivos do Projeto

- Implementar um **analisador léxico** usando expressões regulares.
- Construir um **analisador sintático** (descendente ou ascendente).
- Implementar **verificações semânticas básicas**.
- Criar e manipular uma **tabela de símbolos**.
- Gerar **código C** equivalente ao programa em Portugol.
- Integrar tudo em um **compilador funcional**.

Linguagem Portugol Simplificada (Subset)

O compilador deve reconhecer:

1. Declarações de variáveis

Portugol

```
inteiro x;
real y;
cadeia nome;
```

C

```
int x;
float y;
char nome[100];
```

2. Atribuições de variáveis

Portugol

```
x = 10;
nome = "Maria";
```

C

```
x = 10;
strcpy(nome, "Maria");
```

3. Estrutura Condicional Simples (SE / SENÃO)

Portugol

```
se (x > 10) entao
    x = x + 1;
senao
    x = 0;
fimse
```

C

```
if (x > 10) {
    x = x + 1;
} else {
    x = 0;
}
```

4. Estrutura de Repetição (ENQUANTO)

Portugol

```
enquanto (x < 5) faca
    x = x + 1;
fimenquanto
```

C

```
while (x < 5) {
    x = x + 1;
}
```

5. Módulos: Funções e Procedimentos

O compilador deve suportar funções (com retorno) e procedimentos (sem retorno).

Procedimento (procedure)

```
procedimento mostrar(x)
inicio
    escreva(x);
fim
```

C

```c
void mostrar(int x) {
    printf("%d", x);
}
```

6. Função (function)

Portugol

```
funcao soma(a, b)
inicio
    retorne a + b;
fim
```

C

```c
int soma(int a, int b) {
    return a + b;
}
```

7 Comando de Saída (escreva)

O compilador deve gerar o printf adequado ao tipo:

Tipo Portugol	Tipo C	printf
inteiro	int	%d
real	float	%f
cadeia	char[]	%s

4. Requisitos Técnicos

O compilador deve conter, obrigatoriamente:

Analisador Léxico

- Implementado com expressões regulares
- Deve identificar tokens e reportar erros léxicos

Analisador Sintático

- Pode ser descendente recursivo ou baseado em PLY
- Deve validar a estrutura do programa

Analisador Semântico

- Verificar:
- variáveis declaradas antes do uso
- compatibilidade de tipos
- quantidade e tipos de parâmetros em funções/procedimentos

Tabela de Símbolos

- Deve armazenar:
- nome da variável
- tipo
- escopo
- parâmetros de funções/procedimentos

Geração de Código C

- O código gerado deve ser compilável com gcc ou clang
- Deve conter:

```c
#include <stdio.h>
#include <string.h>

int main() {
    // código traduzido
    return 0;
}
```

Etapas do Compilador

1. Análise Léxica (Lexer)

Usar **re** para reconhecer tokens:

- IDENTIFICADOR
- NUM_INT
- NUM_REAL
- STRING
- PALAVRAS-CHAVE (inteiro, real, se, entao...)
- OPERADORES (+, -, *, /, >, =, ==)
- DELIMITADORES (; , ( ))

O lexer deve gerar pares:

```
(token, lexema)
```

2. Análise Sintática (Parser)

Pode ser:

- **Descendente recursivo**
- Ou usando **PLY (Python Lex-Yacc)**

Exemplo de gramática:

```
programa → lista_comandos
lista_comandos → comando lista_comandos | ε
comando → declaracao | atribuicao | se | enquanto | escreva
```

3. Análise Semântica

Verificar:

- Variáveis declaradas antes do uso
- Tipos compatíveis em atribuições
- Tipos compatíveis em expressões

4. Tabela de Símbolos

Estrutura sugerida:

```
{
   "x": {"tipo": "inteiro"},
   "nome": {"tipo": "cadeia"}
}
```

5. Geração de Código C

Exemplos:

Portugol

```
inteiro x;
x = 10;
escreva(x);
```

C gerado

```c
int x;
x = 10;
printf("%d", x);
```

Execução no Google Colab

O arquivo main.ipynb deve conter:

1. Upload do código Portugol

```
codigo = """
inteiro x;
x = 10;
escreva(x);
"""
```

2. Chamada das etapas

```
from lexer import Lexer
from parser import Parser
from gerador_c import GeradorC

tokens = Lexer().tokenizar(codigo)
arvore = Parser(tokens).parse()
codigo_c = GeradorC().gerar(arvore)

print(codigo_c)
```

Entrega do Projeto

Deve entregar:

Notebook Google Colab com:

- Explicação das etapas
- Código do compilador
- Exemplos de entrada e saída

Arquivos .py organizados

- Um vídeo curto demonstrando o compilador funcionando