

# Projeto Final: Compilador para a Linguagem C–

Prof. Dr. Rodrigo Colnago Contreras

[contreras@unifesp.br](mailto:contreras@unifesp.br)

[Lattes](#) [Google Scholar](#) [ORCID](#)

Criada em 22/10/2025.

## Formação das equipes

O projeto deverá ser desenvolvido em **duplas** ou, no máximo, em **trios**. Todos os membros devem participar do desenvolvimento e serão avaliados também **individualmente** na apresentação, respondendo a perguntas sobre *todas* as partes do compilador.

## Objetivo

Implementar um compilador funcional para a linguagem didática C– (conforme o projeto do Apêndice A do livro de Kenneth C. Louden), contemplando:

- **Scanner** (analisador léxico);
- **Tabela de Símbolos**;
- **Parser** (analisador sintático) e construção da **AST**;
- **Analizador semântico**;
- **Gerador de código intermediário** em três endereços (sem otimizações).

A implementação deve ser em **C** ou **C++**. O uso de **Flex** (*scanner*) e **Bison** (*parser*) é recomendado, mas não obrigatório.

## Requisitos mínimos por módulo

### 1. Scanner (Léxico)

Ler o código-fonte e produzir a sequência de *tokens* com numeração de linhas. Deve reconhecer identificadores, palavras-chave, números, operadores e delimitadores da linguagem C–. Em caso de erro léxico, **interromper** a compilação exibindo:

ERRO LEXICO: ‘‘lexema’’ - LINHA: X
------------------------------------

## 2. Tabela de Símbolos

Estrutura central para a análise semântica. Deve registrar, no mínimo:

- **nome** do identificador;
- **tipo** (`int` ou `void`, conforme C–);
- **escopo** (função/bloco onde foi declarado).

A implementação pode usar *hash table* com pilha de escopos, listas encadeadas ou equivalente.

## 3. Parser (Sintático) & AST

Reconhecer a estrutura gramatical da linguagem e construir uma **Árvore Sintática Abstrata (AST)**. Em erro sintático, exibir mensagem e realizar a melhor *recuperação* possível:

```
ERRO SINTATICO: token inesperado ‘‘token’’, esperado ‘‘token’’ - LINHA: X
```

## 4. Analisador Semântico

Percorrer a AST verificando:

- uso de **variáveis não declaradas**;
- **declarações duplicadas** no mesmo escopo;
- **incompatibilidades de tipo** em expressões/atribuições;
- regras mínimas para `main()`, `input()` e `output()`.

Mensagens de erro:

```
ERRO SEMANTICO: identificador ‘‘id’’ - LINHA: X
```

## 5. Código Intermediário (3 endereços)

Linearizar a AST e produzir código estilo três endereços (sem otimização), por exemplo:

```
t1 = a + b  
t2 = t1 * c  
output t2
```

## Entrada/Saída do compilador

Ao final da compilação, imprimir:

1. **Tabela de Símbolos** completa;
2. **AST** em formato textual (ou arquivo `.dot`/Graphviz);
3. **Código Intermediário** (3 endereços).

## Apresentação e avaliação

O compilador será executado com diferentes programas C – fornecidos pelo docente. A apresentação é **obrigatória** e cada integrante explicará decisões de projeto e trechos de código. Caso um integrante não se mostre integrado ao projeto, todos os integrantes receberão nota 0.

### Critérios e pesos (10,0 pts):

Implementação correta do scanner (léxico)	2,0
Implementação correta do parser (sintático) & AST	2,0
Implementação correta do analisador semântico	2,0
Mensagens de erro (correção e clareza)	1,0
Estrutura/clareza do código & Tabela de Símbolos	1,0
Apresentação e domínio do projeto	2,0
<b>Total</b>	<b>10,0</b>

## Entregáveis

- Relatório com documentação do código-fonte e detalhando a participação de cada membro do grupo.
- Exemplos de uso do material desenvolvido.
- Código-fonte completo (C/C++), com README e instruções de compilação/execução.
- Makefile (ou script equivalente).
- Saídas geradas: Tabela de Símbolos, AST e Código Intermediário.

## Sugestões de ferramentas

**Flex** (*scanner*), **Bison** (*parser*), **Graphviz** (AST), **GCC/Clang/MinGW** (compilação), sistema operacional **Windows**.

## Observações finais

É permitido usar o esqueleto do projeto de C– (estrutura de arquivos) como referência didática, desde que **todo o código seja de autoria do grupo**. Plágio implica reprovação.

## Bibliografia

- LOUDEN, Kenneth C. *Compiler Construction: Principles and Practice*. Boston: PWS Publishing Company, 1997. 707 p.