UNIVERSIDADE FEDERAL DE LAVRAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Curso: *Ciência da Computação*	*GCC130 – Compiladores*	Professor: *Ricardo Terra*
Pontuação: *40 pontos* (3 *questões)*	**TP**	Data: *Veja etapa*

INFORMAÇÕES SOBRE TP:

1. Atividades entregues **após o prazo** terão penalização na nota. Logo, fiquem atentos à data de entrega.
2. Cópias (total ou parcial) serão penalizadas com **nota zero** em todos os trabalhos.
3. A atividade é grupo de 3 (três) alunos.
4. O trabalho deve ser entregue pelo *Campos Virtual* e será avaliado junto com os alunos no laboratório em data estipulada.
5. Envie arquivos *somente* nos formatos *txt* e *pdf* (não enviar *doc*, *docx*, *odt*, etc.). Arquivos compactados somente *zip* (não enviar *tar.gz*, *rar*, *z*, etc.). Não use acentos e "ç" nos nomes de arquivo.

O objetivo do trabalho prático é desenvolver um compilador para um mini C didático. O trabalho está dividido em 3 etapas, conforme descrição a seguir.

1. (5 pts) **Entrega 1 — Analisador Léxico (Flex)**

   Implemente um analisador léxico em **Flex** para a linguagem com os seguintes elementos:

   - Declarações de tipos (`int`, `bool`);
   - Identificadores e números inteiros;
   - Palavras-chave (`if`, `else`, `while`, `print`, `read`, `true`, `false`);
   - Operadores relacionais, aritméticos e lógicos;
   - Símbolos de pontuação (`; , ( ) { }`).

   PS: tipo `int` pode assumir valor negativos.

   Critérios a serem avaliados:

   - Além de reconhecer os *tokens*, deve:
     - estar organizado, com comentários explicativos;
     - imprimir o *token*, seu lexema e sua posição (linha e coluna);
     - detectar erros e reportá-los ao usuário informando a posição;
     - exibir a **tabela de símbolos** construída pelo analisador léxico; e
     - desconsiderar espaços em branco e comentários (linha única `//` e múltipla `/* */`).
   - O **relatório** (conciso e objetivo) deve:
     - discutir as **decisões de projeto**;
     - discutir as **dificuldades encontradas**;
     - apresentar dois **diagramas de transição** (DFAs) referentes a classes de *tokens*; e
     - incluir um arquivo de teste e sua saída (não precisa ser grande, porém completo).

2. (10 pts) **Entrega 2 — Analisador Sintático (Bison)**

   Implemente um analisador sintático em **Bison** para uma gramática que deve prover:

   - declarações de variáveis e tipos;
   - instruções de atribuição;
   - condicionais (`if/else`);
   - laços (`while`);
   - blocos e comandos compostos ({ });
   - instruções de entrada/saída (`print`, `read`);
   - expressões aritméticas, relacionais e lógicas, com a precedência definida; e
   - comentários de uma linha e múltiplas linhas.

   Critérios a serem avaliados:

   - Correções apontadas pelo professor na Etapa 1.

- O analisador deve:
  - estar organizado, com comentários explicativos;
  - reconhecer a sintaxe de programas escritos na linguagem proposta;
  - compilar corretamente no Bison e rodar em conjunto com o analisador léxico da Etapa 1;
  - detectar e reportar **erros sintáticos** com mensagem clara indicando a posição (linha e coluna); e
  - resolver ambiguidades (e.g., *dangling else* e `-`).
- O **relatório** (conciso e objetivo) deve:
  - apresentar a **gramática completa** em BNF da linguagem implementada;
  - discutir as **decisões de projeto** (e.g., ambiguidades e tratamento de erros);
  - discutir as **dificuldades encontradas**;
  - calcular os conjuntos **FIRST** e **FOLLOW** apenas para os não-terminais de **expressões**;
  - apresentar e comentar os estados do **autômato LR(0)** com conflitos (e.g., *dangling else* e `-`); e
  - incluir um arquivo de teste e sua saída (não precisa ser grande, porém completo).

3. (25 pts) **Entrega 3 — Análise Semântica e Geração de Código Intermediário**

   Implemente a **análise semântica** e a **geração de código de três endereços (IR)** para a mesma linguagem das Etapas 1 e 2.

   Critérios a serem avaliados:

   - Correções apontadas pelo professor na Etapa 2.
   - O compilador deve:
     - manter uma **tabela de símbolos** com **escopos aninhados** (abrir/fechar escopo em { }), armazenando ao menos *identificador*, *tipo* e *nível de escopo*;
     - realizar **verificação de tipos** em:
       - **atribuição** (lado direito compatível com o tipo da variável);
       - **expressões aritméticas** (`+ - * / %` aceitam apenas `int`);
       - **relacionais** (`== != < <= > >=` tomam `int` e produzem `bool`);
       - **lógicos** (`! && ||` aceitam apenas `bool`);
       - **unários** (`-` unário sobre `int`, `!` sobre `bool`); e
       - `if`/`while`: condição deve ser `bool`.
     - **reportar erros semânticos** com posição, e.g., identificadores não declarado ou redeclarados no mesmo escopo.
     - gerar código intermediário para todas as construções da linguagem;
       - usar de temporários $(t_1, t_2, \ldots, t_n)$ e rótulos $(L_1, L_2, \ldots, L_m)$; e
       - especificar e usar um conjunto mínimo de instruções de IR.
   - O **relatório** (conciso e objetivo) deve:
     - descrever a **estrutura da tabela de símbolos** e o **gerenciamento de escopos**;
     - apresentar uma **tabela de regras de tipagem** dos operadores/construções;
     - explicar a **estratégia de geração de IR** para `if/else` e `while`;
     - incluir um **programa de teste** e o **IR gerado** correspondente (saída do compilador);
     - apresentar pelo menos uma **Tradução Dirigida por Sintaxe (TDS)**; e
     - discutir **decisões de projeto** e **dificuldades encontradas**.