# Compiladores

prof. Ricardo Oliveira

2024/1 - **Trabalho da disciplina**

## Enunciado

**Invente** uma nova linguagem de programação, e implemente um **compilador** que compila programas da sua linguagem para o *assembly* do *Raposeitor* (isto é, implemente um programa que, **dado um programa** escrito na sua linguagem de programação inventada, **gera o código equivalente em *assembly*** do *Raposeitor*).

Com a linguagem de programação que você inventar, deve ser possível:

- declarar e manipular variáveis inteiras;
- declarar e manipular vetores (estáticos) de inteiros;
- ler valores (do usuário) e imprimir valores (para o usuário);
- imprimir textos/strings (para o usuário);
- escrever seleções completas (equivalente a "if"/"else");
- escrever seleções parciais (equivalente a "if" sem "else");
- escrever laços com quaisquer condições (equivalente a "while");
- escrever laços simplificados de repetição (equivalente a "for");
- aninhar laços e seleções;
- escrever e usar expressões aritméticas (com soma, subtração, multiplicação, divisão inteira, resto da divisão), expressões de comparação (com maior, menor, maior-ou-igual, menor-ou-igual, igual e diferente) e expressões lógicas (com "não", "e" e "ou"), incluindo as com parênteses;
- escrever comentários de linha (o equivalente a "//").

A sua linguagem não precisa fornecer: variáveis não inteiras (float/char/etc); estruturas de dados além do vetor (matriz/struct/etc); ponteiros/endereçamento de memória; funções; arquivos; demais elementos não listados acima.

Você tem liberdade para inventar sua linguagem de programação como quiser! *Apenas como exemplo*, os seguintes programas, escritos em linguagens *inventadas*, lêem dois valores e imprimem o maior deles:

```
var a: inteiro;
var b: inteiro;
var maior: inteiro;     # declaracao

leia(a);                # leitura
leia(b);
se a > b entao {        # teste
    maior <- a;
} senao {
    maior <- b;
}

print("O maior eh ", maior, "\n");
```

```
DECLARA a AI
DECLARA b AI
DECLARA maior AI                    @ declaracao

MEDAH a AI                          @ leitura
MEDAH b AI
SERAQUE a EHMAIORQUE b ? SEFOR (    @ teste
    maior RECEBA a AI
) SEINAO (
    maior RECEBA b AI
)

MOSTRA "O maior eh" DEPOIS maior DEPOIS "\n" AI
```

Veja uma outra linguagem inventada que ficou famosa: `https://birl-language.github.io/` .

Use a criatividade!

# Raposeitor

Neste trabalho, você deve implementar um compilador que, dado programas escritos na linguagem que você inventou, gera seu código equivalente em *assembly* do *Raposeitor*. O *Raposeitor* é uma máquina simplificada cujo conjunto de instruções segue códigos de três endereços. **A documentação completa e a resumida do *Raposeitor* estão no Moodle**.

Como exemplo, o pacote em anexo ao trabalho contém vários problemas escritos em Rapolang – linguagem inventada como exemplo – e os códigos em *assembly* do *Raposeitor* que um compilador dessa linguagem geraria para cada um deles. Mas atenção! Você não deve fazer um compilador de Rapolang, mas sim da linguagem de programação que você inventou!

Você pode executar os códigos em *assembly* do Raposeitor gerados pelo seu compilador com os interpretadores dados, conforme a documentação.

# Implementação

- Seu compilador deve ser implementado em C ou em C++;
- Você pode usar qualquer técnica de *parser* que preferir na sua implementação;
- Você pode implementar o compilador manualmente, ou usar geradores de analisadores como as ferramentas flex/bison;
- Você pode restringir alguns elementos da sua linguagem inventada para facilitar a implementação (por exemplo: *obrigar* que variáveis declaradas sejam inicializadas; que blocos tenham delimitadores (como `{` e `}` em C/C++) mesmo que só tenham uma única linha de código; etc);
- Seu compilador deve apontar a ocorrência de erros (léxicos, sintáticos ou semânticos) em programas que não compilam. Neste caso, seu compilador deve indicar o tipo do erro (se o erro é léxico, sintático ou semântico), e descrever brevemente o erro em si;
- O código *assembly* gerado pelo seu compilador não precisa ser otimizado, bastando que funcione corretamente com os interpretadores dados (`raposeitor.cpp` e `raposeitor.py`).

## Orientações

- O trabalho pode ser feito por equipes de *no máximo* 3 (três) estudantes;
- O trabalho consiste na implementação de quatro partes: análise léxica; análise sintática; análise semântica e geração de código. O trabalho deverá ser desenvolvido durante todo o semestre, e será avaliado em dois momentos, recebendo uma nota para cada momento:
  - Em um *checkpoint* após a implementação da análise léxica e sintática. Neste ponto, o trabalho (parcial) deverá ser entregue, e seu desenvolvimento deverá ser *apresentado* à turma em sala de aula;
  - Com o trabalho completo. Neste ponto, o trabalho final deverá ser entregue, e seu desenvolvimento deverá ser *defendido* para o professor.
- Em *ambos* os casos, submeta, via *Moodle*, um pacote **zip** ou **tar.gz** contendo:
  - todo o código-fonte necessário para compilar e executar sua solução;
  - um arquivo de texto (txt) onde conste:
    - O nome de todos os integrantes da equipe;
    - Toda informação que a equipe julgar relevante para a correção (como *bugs* conhecidos, detalhes de implementação, escolhas de projeto, etc.)
  - programas de exemplo escritos na linguagem inventada. Inclua, *pelo menos*, os seguintes programas:
    - lê três números e imprime o maior;
    - imprime todos os números da sequência de Fibonnaci até um dado valor;
    - le um vetor do usuário e o ordena pelo *InsertionSort*.

    No trabalho final (versão completa), deve ser possível compilar e gerar o código *assembly* respectivo para cada programa dado.
- A equipe deverá estar completa tanto na apresentação quanto na defesa;
- Durante as apresentações e defesas, a equipe poderá ser solicitada a escrever outros programas na linguagem inventada, além dos citados acima. Durante a correção, outros programas também poderão ser escritos e testados pelo professor;
- A defesa do trabalho final faz parte da avaliação. Durante a defesa, o professor poderá fazer perguntas a toda a equipe ou individualmente, e poderá atribuir notas diferentes a membros de uma mesma equipe;
- Comente adequadamente seus códigos para facilitar a correção;
- O trabalho deve ser entregue até **5 de Maio de 2023, 23:59** (*Checkpoint*) e até **16 de Junho de 2023, 23:59** (versão final), apenas via *Moodle*. Trabalhos entregues por outros meios ou fora do prazo não serão aceitos. É suficiente que o trabalho seja submetido por apenas um estudante da equipe;
- A equipe que entregar e apresentar o trabalho no *checkpoint* deve ser **a mesma** a entregar e defender o trabalho final;
- É permitido que equipes que não apresentarem o trabalho parcial enviem e defendam o trabalho final (neste caso, apenas a nota referente ao trabalho final será considerada na média);
- Trabalhos detectados como cópia/plágio (de colegas, da internet ou de ferramentas de IA) ou comprados receberão **todos** a nota 0 (**ZERO**) e estarão sujeitos a abertura de Processo Administrativo Disciplinar Discente.