# MC202 - **Estruturas de Dados**

## Lab 09 - Árvores Binárias de Busca

**Data da Primeira Chance:** 16 de outubro de 2023
**Link da atividade:** https://classroom.github.com/a/6RXu1rVw
**Peso:** 4

O processo de compilação de um código é composto de diversas fases até chegar no código de máquina, as instruções em binário. Desde análise léxica até otimizações, seu compilador da sua linguagem favorita precisa fazer o melhor para transformar o seu laboratório de MC202 em um conjunto de instruções que um computador consegue utilizar.

Dito isso, **Daenerys** Targaryen (A Primeira de seu Nome, Nascida da Tormenta, A Não Queimada, Mãe de Dragões, Khaleesi do Grande Mar de Grama, Quebradora de Correntes, Rainha de Mereen, Rainha dos Ândalos, dos Roinares e dos Primeiros Homens, Senhora dos Sete Reinos e Protetora do Reino) resolveu se modernizar e está aprendendo Estrutura de Dados. Em Westeros, a linguagem de programação mais famosa é **D** (dracarys), muito similar com o nosso C.

Infelizmente, nossa rainha está tendo muitas dificuldades. Ela já tomou todo tipo de erro: *segmentation fault*, vazamento de memória, leitura inválida… Ela também já botou a culpa em todos: no computador dela, no **GoT**hub (sistema de submissão que o professor dela usa) e no compilador.

A mais nova suspeita dela é que o compilador que ela usa (**johncc**) está com problemas e não consegue identificar erros de tipos inválidos. Assim, operações como:

```
int a = 1;
int b;
char c = a + b; // tipos incompatíveis
```

passam despercebidas pelo compilador e dão problema na execução.

Nesse laboratório, sua função é ajudar a Nascida da Tormenta a conferir seu código antes de compilar. Para isso, você irá implementar uma Tabela de Símbolos, uma das estruturas que um compilador possui para conferir a semântica dos códigos.

Uma tabela de símbolos é uma estrutura de dados que guarda um conjunto de símbolos e informações sobre estes. Para simplificar as coisas neste momento, um símbolo é um nome de

uma variável. Um compilador como o gcc e johncc guardam muitas informações, mas para efeitos do problema da Senhora dos Sete Reinos, precisamos apenas do tipo que uma variável possui.

Para agilizar as consultas de informações, você deve implementar uma Árvore de Busca Binária. Para os casos maiores, será necessário utilizar alguma técnica de balanceamento.

### Informações sobre a linguagem **D:**

- Uma declaração sempre começa com o tipo da variável (int, double ou char) seguido do nome de uma única variável. Ou seja: `tipo simbolo;`
- Não é possível ter mais de uma declaração por linha.
- Só é necessário conferir tipos e/ou existência de variáveis em linhas onde ocorrem operações aritméticas (soma, subtração, multiplicação, divisão e resto) entre duas variáveis e atribuem o resultado em outra:
  - `var1 = var2 + var3;`
- Atente-se que o tipo do resultado de uma operação depende do tipo das variáveis e da operação em si. Neste aspecto, a linguagem D é igual a C.
- Existem todas as operações aritméticas para todos os tipos (não me pergunte o que significa a divisão de dois chars, *"I know nothing"*).
- Não há "casting" (mudança de tipo) automática. Operações aritméticas devem ser sempre feitas entre tipos iguais.
  - Sejam `a` e `b` inteiros e `c` um double:
  - `c = a + b; // erro!`
  - `a = a + c; // erro!`

# Entrada

A entrada do programa será um programa escrito em **D**.

O fim da entrada ocorre quando o arquivo terminar, ou seja, no End of File (EOF). Mais sobre isso na seção de dicas.

Uma linha do programa pode conter, no máximo, 200 caracteres.

# Saída

`SIMBOLO1 = SIMBOLO2 OP SIMBOLO3`

Para cada linha que contém uma operação aritmética, como a da linha acima, você deve conferir:

1. Se os três símbolos usados estão definidos.
2. Se os tipos do segundo símbolo e terceiro são diferentes para serem operados.
3. Se o resultado da operação possui o mesmo tipo que a variável que o receberá.

Caso ocorra o primeiro erro, você deve imprimir a seguinte mensagem de erro:
`Linha N: o símbolo SIMBOLO não foi definido.`

Caso ocorra o segundo erro, você deve imprimir a seguinte mensagem de erro:
`Linha N: tipos incompatíveis: SIMBOLO2 (TIPO2) OP SIMBOLO3 (TIPO3).`

Caso ocorra o terceiro erro, você deve imprimir a seguinte mensagem de erro:
`Linha N: tipos incompatíveis: o símbolo SIMBOLO1 espera TIPO1, obteve TIPO_RESULTADO.`
onde TIPO_RESULTADO é o tipo do resultado da operação.

Caso haja mais de um erro em uma linha, o primeiro erro descrito aqui deve ser o impresso. Caso haja o mesmo erro para mais de uma variável, a primeira variável da linha com erro deve ser a impressa no erro. Não se deve imprimir mais de um erro em uma linha.

Caso não haja nenhum erro no código, seu programa deve imprimir:
`Não há erros de tipo.`

Ao final da saída, você deve imprimir os símbolos declarados em ordem alfabética, seguindo o seguinte formato:
`simbolo tipo`

# Exemplos

## Exemplo 1:

**Entrada**

```
#include <lanister.h>

int main () {
    int a;
    int b;

    c = b + a;
    printf("%d", c);

    char d;
    a = d + b;

    d = b + a;

    return 0;
}
```

**Saída**

```
Linha 7: o símbolo c não foi definido.
Linha 11: tipos incompatíveis: d (char) + b (int).
Linha 13: tipos incompatíveis: o símbolo d espera char, obteve int.
a int
b int
d char
```

## Exemplo 2:

**Entrada**

```
#include <westeros.h>

int main() {

      int Addison_Hoover;
      int Claire_Hancock;
      char Enrique_Cervantes;
      char Luke_Mcneil;
      Addison_Hoover = Addison_Hoover + Claire_Hancock;

      return 0;
}
```

**Saída**

```
Não há erros de tipo.
Addison_Hoover int
Claire_Hancock int
Enrique_Cervantes char
Luke_Mcneil char
```

# Dicas

Perceba que toda operação deve ser feita entre duas variáveis que possuem o mesmo tipo.
Existirão diversas linhas que você não precisará conferir, mas precisará contar.
Para a leitura da entrada, você pode usar o `fgets` para ler uma linha e `sscanf` para ler palavras nela, como no exemplo abaixo:

```
char buffer[MAX];
```

```
char simb1[MAX];
fgets(buffer, MAX, stdin);
sscanf(buffer, "%s", simb1);
```

Um detalhe a ser observado, diferente do `scanf` ao ler o `stdin`, o `sscanf` lendo de uma string não percorre a string. Ou seja, `sscanf` consecutivos vão sempre começar do começo do buffer. Assim como o `scanf`, o `sscanf` retorna a quantidade de variáveis lidas com sucesso.

O fgets pode ser usado para ler as linhas até encontrar o final do arquivo (EOF). Você pode conferir se ele o encontrou usando:

```
if (feof(stdin)) {
  break;
}
```

Sobre quais linhas ignorar e como achar as declarações e expressões aritméticas:

- Toda linha que contém um parêntese pode ser ignorada (eles não são usados em expressões).
- Toda linha cuja primeira palavra não começa com uma letra, pode ser ignorada. Perceba que o `fgets` irá pegar os espaços de indentação, mas o `sscanf("%s")` não.
- Toda linha (que ainda não caiu nos casos anteriores) que começa com um tipo, é uma declaração.
- Toda linha (que ainda não caiu nos casos anteriores) e que possui o formato abaixo é uma expressão.
    - `SIMBOLO1 = SIMBOLO2 OP SIMBOLO3`

# Regras e Avaliação

Seu código será avaliado não apenas pelos testes do GitHub, mas também pela qualidade. Dentre os critérios subjetivos de qualidade de código analisaremos neste laboratório:

- A escolha de bons nomes de funções e variáveis;
- A ausência de trechos de código repetidos desnecessariamente;
- O uso apropriado de funções;
- A ausência de vazamentos de memória;
- A eficiência dos algoritmos propostos;
- A correta utilização das Estruturas de Dados;
- Você deve implementar uma árvore de busca binária para recuperar informações quando for necessário.

Note, porém, que essa não é uma lista exaustiva, pois outros critérios podem ser analisados dependendo do código apresentado visando mostrar ao aluno como o código poderia ser melhor.

# Submissão

Você deverá submeter no repositório criado no aceite da tarefa. Você pode enviar arquivos adicionais caso deseje para serem incluídos por `dracarys.c`. Não se esqueça de dar `git push` !

**Atenção:** O repositório da sua atividade conterá alguns arquivos iniciais. Fica **estritamente proibido** ao aluno alterar os arquivos já existentes, tais como o testador existente ou demais arquivos de configuração do laboratório.

Lembre-se que sua atividade será corrigida automaticamente na aba “Actions” do repositório. Confirme a correção e o resultado, já que o que vale é o que está lá e não na sua máquina.

Após a correção da primeira entrega, será aberta uma segunda chance, com prazo de entrega apropriado.