

# Aula Prática 6

Lucas Albano Olive Cruz - 2022036209

## 1. Avaliação Qualitativa

O programa recebe uma expressão e a armazena em uma árvore, antes disso ele executa uma série de verificações. Para cada verificação são feitas diversas comparações e na fase de inserção diversas escritas. Ainda, são feitas diversas chamadas recursivas ao caminhar pela árvore e resolver a expressão. Estas partes devem ser as mais custosas do programa.

## 2. Plano de caracterização Localidade Referência

Na fase de verificação várias sequências de localidades provavelmente serão acessadas pois a expressão é lida de forma contígua. Após isso haverá uma transição para a parte de inserção na quais partes não necessariamente contíguas da memória serão acessadas pois a árvore é montada de forma dinâmica.

## 3. Parâmetros Caracterizados

Portanto, serão analisados os parâmetros de acessos de memória (leitura e escrita) nas fases de validação, inserção, caminhamento e resolução da expressão na árvore sintática. Além do número de chamadas das principais funções do programa.

Será usada uma entrada de tamanho médio seguida de operações de conversão e resolução.

## 4. Execução com Cachegrind

```
-----  
Profile data file 'cachegrind.out.5212'  
-----
```

```
Il cache:      32768 B, 64 B, 8-way associative  
Dl cache:      32768 B, 64 B, 8-way associative  
LL cache:      2097152 B, 64 B, 8-way associative  
Profiled target: ./bin/programa  
Events recorded: Ir Ilmr ILmr Dr Dlmr DLmr Dw Dlmw DLmw  
Events shown:   Ir Ilmr ILmr Dr Dlmr DLmr Dw Dlmw DLmw  
Event sort order: Ir Ilmr ILmr Dr Dlmr DLmr Dw Dlmw DLmw  
Thresholds:     99 0 0 0 0 0 0 0 0  
Include dirs:  
User annotated:  
Auto-annotation: on  
-----
```

The call graph illustrates the execution flow of the program. It begins with a 'below main' node, which leads to the 'main' function. From 'main', the flow branches into several other functions, including 'std::ios\_base::init', 'std::basic\_istream<char, std::char\_traits<char> >::getline', and 'ArvoreBinaria::CaminhoInt'. The graph continues to show various standard library functions and custom functions, ending with 'float \_\_gnu\_cxx::\_\_stack\_float, float, char (<float\*>)[char const\*, char\*\*], char const\*'. The graph is color-coded and includes call counts for each edge.

Assim é evidente que a fase de validação da expressão é altamente custosa por necessitar de várias instruções de leituras para comparar os caracteres da expressão. Ainda é interessante notar que ambos métodos de caminhamento gastam a mesma quantidade de leituras pois percorrem a mesma árvore sintática, no entanto o caminhamento `inOrdem` gasta mais escritas por necessitar de adicionar parênteses.

Agora analisando o `callgrind` também possuímos algumas informações interessantes. A função `validaExpressao` faz 358 chamadas a funções `std` de comparação, novamente, percebemos ser um método muito custoso ao programa. O caminhamento `inOrdem` faz 50 chamadas recursivas a ele mesmo e 74 chamadas ao `ostream` para escrever no terminal. O método `resolveRecursivo` faz 76 chamadas recursivas e 20 chamadas a função `stof` para converter uma string para um float.

Assim, creio que exceto a fase de validação, o programa se comporta relativamente bem em relação aos gastos de memória. A estrutura de dados árvore é bastante eficiente para armazenar e possibilitar acesso à expressão.