IMD0030 – LINGUAGEM DE PROGRAMAÇÃO I

Aula 04 – Profiling (**gprof**)

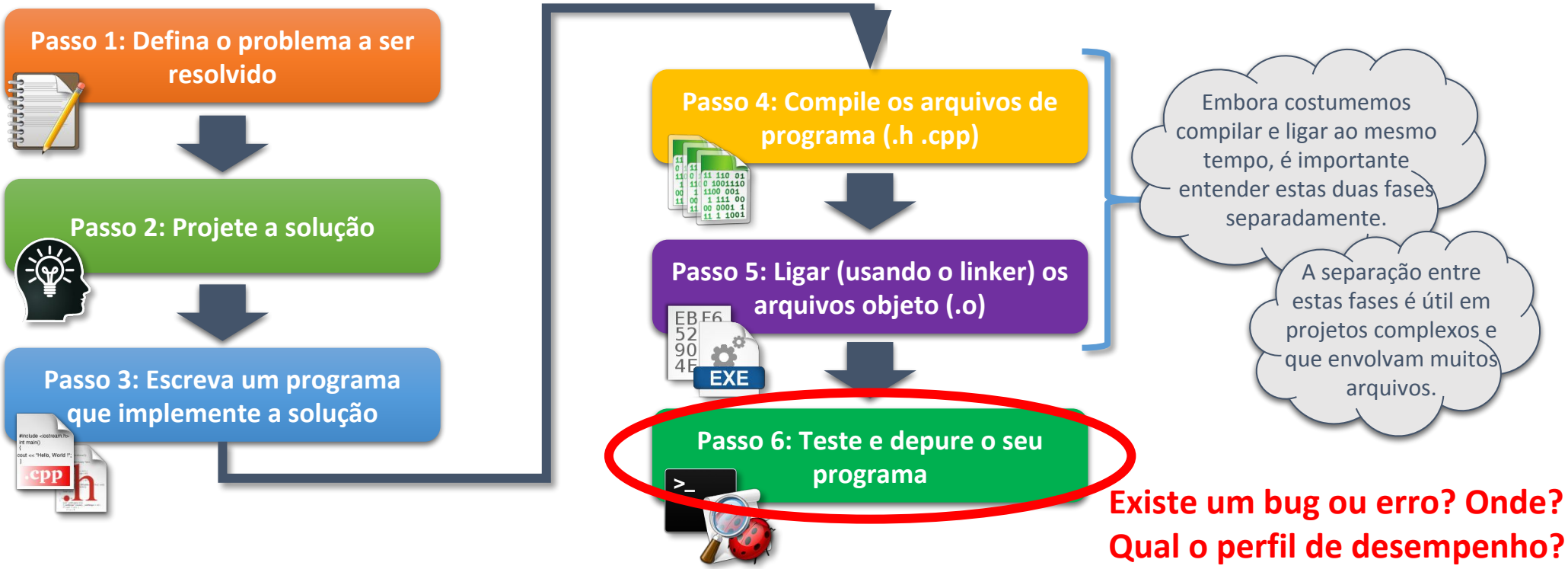
(material baseado nas notas de aula do Prof. Silvio Sampaio e Prof. César Rennó-Costa)

INSTITUTO METRÓPOLE DIGITAL

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Introdução

- Relembrando…



Profiling

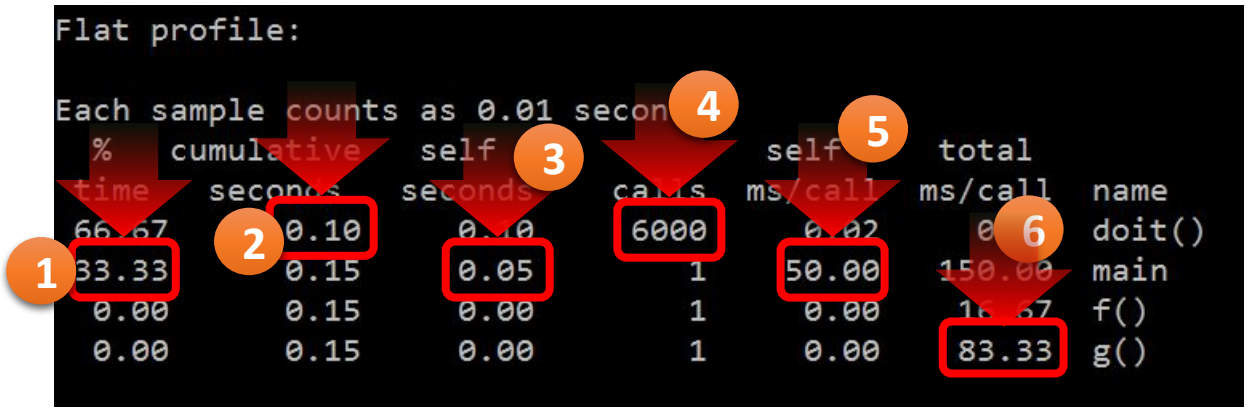
- Um **profiler** é um programa utilizado para avaliar o desempenho do seu programa, permitindo encontrar os gargalos (pontos onde o programa demora mais)
  - Apresenta, entre outras informações, um gráfico com o tempo de execução de cada função
  - Essa ferramenta também permite conhecer as funções presentes no código, o número de chamadas dentro de cada função e a porcentagem de tempo gasto em cada uma delas
- O **gprof** (GNU Profiling) é uma ferramenta que faz parte do GCC (GNU Compiler Collection), desenvolvida por Jay Fenlason, que serve para medir o tempo gastos pelas funções de um algoritmo, e exibi-las
  - Originalmente escrito por um grupo liderado por Susan L. Graham na University of California, Berkeley para o Berkeley Unix (4.2BSD)

Gprof

- Para usar o **gprof**
  - Primeiro deve-se compilar o programa incluindo a diretiva de compliação **-pg**, que fará com que o compilador insira informações adicionais para o profiler
  - Executar o programa uma vez para que seja criado o profile **gmon.out** que será lido pelo **gprof**
  - Executar o **gprof** sobre o executável
    - Exemplo 1: **$ gprof --brief -p programa**
      - Neste exemplo as informações de profiling serão mostradas na tela
      - Muitas vezes é muita informação na tela (difícil de ler!)
    - Exemplo 2: **$ gprof –brief -p programa > profile.log**
      - Neste exemplo as informações de profiling serão armazenadas no arquivo profile.log

Gprof

- **Flat profile**: **gprof --brief -p <programa>**
  - Exibe informações sobre a quantidade total de tempo gasto pelo programa na execução de cada função
  - Exemplo: **$ gprof --brief -p ./prog**



1. Percentagem do tempo total gasto na execução da função principal (main).
2. Tempo gasto executando esta função e as demais listadas acima desta.
3. Quantidade de segundos contados apenas para esta função.
4. Quantidade de vezes que a função foi invocada.
5. Quantidade média de *ms* gasta por chamada a esta função sozinha.
6. Quantidade média de *ms* gasta por chamada a esta função e suas subrotinas

Gprof

- **Call graph**: **gprof --brief -q <programa>**
  - Exibe informações sobre as chamadas das funções ao longo da execução do programa
    - Facilita encontrar funções que não demandam muito tempo porém chamam funções que demandam muito tempo
  - Exemplo: **$ gprof --brief -q ./prog**
  - Resultados:
    - A função g() foi chamada pela função main() apenas 1/1 vez
    - A função g(), por sua vez, chamou a função doit() 5000/6000 vezes

```
index % time    self  children    called     name
                                               <spontaneous>
[1]    100.0    0.02    0.06                 main [1]
                0.00    0.05       1/1           g() [3]
                0.00    0.01       1/1           f() [4]
-----------------------------------------------
                0.01    0.00    1000/6000        f() [4]
                0.05    0.00    5000/6000        g() [3]
[2]     75.0    0.06    0.00    6000         doit() [2]
-----------------------------------------------
                0.00    0.05       1/1           main [1]
[3]     62.5    0.00    0.05       1         g() [3]
                0.05    0.00    5000/6000        doit() [2]
-----------------------------------------------
                0.00    0.01       1/1           main [1]
[4]     12.5    0.00    0.01       1         f() [4]
                0.01    0.00    1000/6000        doit() [2]
-----------------------------------------------
```

Gprof

- Maiores detalhes sobre o uso do **gprof** podem ser encontrados em:
  - **https://sourceware.org/binutils/docs/gprof/**
- Exemplos de uso do **gprof** podem ser encontrados em:
  - **http://www.thegeekstuff.com/2012/08/gprof-tutorial/**

Gprof na prática

- Programa exemplo:
  - **$ g++ -Wall -pedantic -g -O0 -pg -o exemplo exemplo.cpp**
  - **$ ./exemplo**
    - Cria o arquivo **gmon.out**

- **Qual das funções usadas neste programa consome mais tempo de execução?**

```
#include <iostream>
#include <cmath>

#define MAX 10000

void doit()
{
   double x=0;
   for (int i=0;i<MAX;i++) x+=sin(i);
}
void f() { for (int i=0;i<1000;++i) doit();}
void g() { for (int i=0;i<5000;++i) doit();}

int main(void)
{
   double s=0;
   for(int i=0;i<1000*MAX;i++) s+=sqrt(i);
   f();
   g();
   std::cout << "Done"<< std::endl;
   return 0;
}
```

?

Aula 04

Profiling (gprof)

Aula 05

Recursividade