



## Laboratórios de Programação

### Guia para Aula Laboratorial 6

Licenciatura em Engenharia Informática

#### Sumário

Compilação e execução de programas em C, Java e Python. Introdução aos Ambientes de Desenvolvimento Integrados. Depuração e perfilagem de programas.

## Programming Laboratories

### Guide for Laboratory Class 6

Degree in Computer Science and Engineering

#### Summary

Compilation and execution of C, Java, and Python programs. Introduction to Integrated Development Environments. Program debugging and profiling.

#### Pré-requisitos:

Algumas das tarefas propostas a seguir requerem o acesso à Internet e acesso a um **sistema operativo Linux**. Se não pretender instalar uma distribuição de Linux na sua máquina pode sempre optar por instalar a distribuição numa máquina virtual. O uso de Subsistema Windows para Linux (*Windows Subsystem for Linux*), para Windows 10, concretiza também uma opção válida.

## 1 Compilação e Execução de Programas na Linguagem de Programação Java

*Compilation and Execution of Programs using the Java Programming Language*

O Java é uma linguagem de programação **orientada a objetos** desenvolvida pela Sun Microsystems, Inc. em 1991. Os programas em Java são compilados para um conjunto de instruções interpretadas por uma máquina virtual (Java *Virtual Machine*, designada por JVM).

#### Tarefa 1 Task 1

Caso seja necessário, instale o Java *Development Kit* (JDK) utilizando a seguinte *hiperligação* ou instalando o pacote `default-jdk`. Como exemplo, em Ubuntu faça: `$ sudo apt install default-jdk`

#### Tarefa 2 Task 2

Crie uma nova diretoria, denominada `Lab_6`. Depois de navegar para a nova diretoria, crie um novo ficheiro, denominado `HelloWorld.java`, com o trecho de código seguinte:

```
class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

```
}
```

Finalmente, compile o ficheiro criado.

**Q1.: Qual foi o comando que utilizou para compilar?**

- ☐ `java HelloWorld.java`
- ☐ `javac HelloWorld.java`
- ☐ `javacc HelloWorld.java`
- ☐ `gcc HelloWorld.java`

Atente agora à diretoria criada na tarefa anterior.

**Q2.: Foi criado algum ficheiro novo, resultante da compilação?**

- ☐ Sim, foi criado um ficheiro denominado `a.out`.
- ☐ Sim, foi criado um ficheiro denominado `HelloWorld.java`.
- ☐ Sim, foi criado um ficheiro denominado `HelloWorld.class`.
- ☐ Não, neste diretoria apenas existe o ficheiro `HelloWorld.java`.

**Q3.: Qual o comando que sugere utilizar para executar o programa que compilou na tarefa anterior?**

- ☐ `java HelloWorld.java`
- ☐ `java HelloWorld`
- ☐ `java HelloWorld.class`

☐ `gcc HelloWorld.java`

## 2 Interpretação e Execução de Programas na Linguagem de Programação Python

*Interpretation and Execution of Programs using the Python Programming Language*

A linguagem de programação Python é uma linguagem interpretada (e não compilada). O interpretador Python (alternativamente, Máquina Virtual Python), escrito em C, é responsável por interpretar o código Python.

Verifique se possui na sua máquina a última versão do interpretador Python utilizando o comando `python3 --version`. Caso não tenha uma versão 3.xx do interpretador de Python instalada deve efetuar a sua instalação utilizando a seguinte *hiperligação* ou instalando o pacote `python3`.

### Q4.: O que pode comentar relativamente ao processo de compilação de um ficheiro Python?

- ☐ É similar a C e Java, em que o código fonte é compilado para um ficheiro *bytecode* e depois executado.
- ☐ O meu comentário é que o processo é bonito, mas demorado.
- ☐ Difere de C e Java, pois o ficheiro com o código fonte não é tipicamente compilado, mas sim interpretado.

### Tarefa 3 Task 3

Crie um ficheiro, denominado `helloWorld.py`, com o trecho de código seguinte:

```
print("Hello , World!")
```

Efetue todos os passos necessários para executar (leia-se *interpretar*) o programa em Python.

### Q5.: Qual o comando que utilizou para executar o programa da tarefa anterior?

- ☐ `java helloWorld.py`
- ☐ `cc helloWorld.py`
- ☐ `python helloWorld.py`
- ☐ `python3 helloWorld.py`

Por omissão, nas distribuições Linux terá também uma versão 2.xx do interpretador Python. Para fazer uso deste interpretador deverá usar o comando `python filename.py`, enquanto

`python3 filename.py` fará uso do interpretador de Python 3.xx.

### Tarefa 4 Task 4

Verifique a diferença de versões dos seus interpretadores `python` e `python3`, usando a opção que achar mais conveniente, e registe os comandos usados.

## 3 Depuração de Programas

### *Program Debugging*

A *depuração* de programas é o processo de localizar e remover erros ou anormalidades (*bugs*) do programa, fazendo uso de ferramentas de depuração.

À medida que os seus programas se tornam cada vez mais complexos será necessário depurar a execução de um programa. Para tal, é muito útil adicionar instruções auxiliares e temporárias para verificação do conteúdo das variáveis, e pontos de paragem ou *break points* ao seu código fonte, que se traduzem em instruções que forcem pausas nos locais assinalados por esses pontos de rutura. Isto é muito útil não só para observar o valores das variáveis em tempo de execução mas também para observar possíveis comportamentos inesperados do programa. Para explorar estas possibilidades, execute as seguintes tarefas.

### Tarefa 5 Task 5

Crie um ficheiro, denominado `Tarefa4.c`, com o trecho de código seguinte:

```
#include <stdio.h>

int f(int a){
    return a-1;
}

int main(){
    int a=1, r, b=f(a);
    r = a/b;
    return 0;
}
```

Compile o ficheiro criado e execute o ficheiro resultante da compilação. Caso queira, pode fazer uso do `Makefile` criado no guia laboratorial anterior e usar o comando `make`.

**Q6.: O programa foi executado com sucesso?**

- ☐ Sim, não vejo nada que indique que tenha havido um erro; por outro lado, eu ainda nem sequer o compilei, por isso...
- ☐ Não, apareceu aqui um erro muito estranho. O *core* foi *dumped*?

**Observe com atenção o código. Q7.: Quais os potenciais pontos onde podem surgir problemas?**

- ☐ Talvez `b=f(a)`, visto estar a usar a variável `a` na mesma linha onde ela foi declarada e instanciada.
- ☐ Diria que na divisão, `r=a/b`. Sempre tive problemas com a matemática e o compilador pode também estar a ter algumas dificuldades. Deduzo que o compilador tirou má nota a matemática...
- ☐ Quase de certeza que o problema vem do `include`.
- ☐ Creio que retornar zero não é a melhor opção. Eu gosto sempre de retornar valores positivos.

**Tarefa 6 Task 6**

Uma abordagem para verificar se o programa chegou a determinada instrução é fazendo *prints* dos valores de variáveis.

Introduza um `printf` para avaliar os valores das variáveis `a` e `b`, na linha imediatamente abaixo da declaração destas, e introduza um `printf`, para avaliar o valor de `r`, na linha imediatamente abaixo da linha da divisão, `r=a/b`.

**Q8.: Quais foram os valores observados das variáveis `a`, `b` e `r`, respetivamente?**

- ☐ Não sei quais são os valores de `a` ou `b` pois o programa encontrou um erro antes deste `printf`.
- ☐ `a=1`, `b=0` mas não sei qual é o valor de `r` pois o programa encontrou um erro antes deste `printf`, que engraçado!
- ☐ `a=1`, `b=1` e `r=1`.

**Q9.: O que pode concluir relativamente à localização do erro no programa?**

- ☐ Este encontra-se na linha de declaração de variáveis.
- ☐ O erro está dentro da função `f`.
- ☐ Há um problema na linha da divisão, `r=a/b`.
- ☐ O programa tem um problema no valor de retorno da função `main`.
- ☐ O erro está em parte incerta, atualmente em fuga e considerado perigoso.

**Tarefa 7 Task 7**

O exemplo apresentado na tarefa 4 era de simplicidade suficiente para que o uso de *prints* de valores de variáveis fosse suficiente para que se pudesse concluir qual a origem do problema no programa. No entanto, para programas de maior dimensão, esta abordagem pode não ser exequível. Neste sentido, existem ferramentas que nos podem auxiliar no processo de depuração de programas.

Obtenha um ficheiro `.c` usando a *hiperligação*. Coloque este ficheiro numa nova diretoria denominada `Lab_6_Debug` Compile o programa usando os comandos necessários ou o `Makefile` criado no guia laboratorial anterior. No final, execute o programa.

**Q10.: O que pode comentar relativamente ao resultado obtido da execução do programa?**

- ☐ O programa foi compilado e executado sem erros e imprimiu o meu número de telemóvel. Como é que isto é possível?
- ☐ O programa foi compilado e executado sem erros mas o resultado é um pouco estranho...
- ☐ Não posso comentar muito pois tive um erro a compilar.

**Q11.: E se executar várias vezes o programa obtém um resultado diferente?**

- ☐ Não. Claramente terá de dar sempre o mesmo resultado.
- ☐ Sim. Mas o programa é confuso e existem ali uns `rand()` no meio. Pode ser essa a razão.
- ☐ Não sei. Continuo sem conseguir compilar e não tenho o `Makefile` do guia laboratorial anterior para me ajudar.

**Tarefa 8 Task 8**

Analise detalhadamente os ficheiros obtidos na tarefa anterior e tente perceber qual o fluxo do programa.

**Q12.: Consegue formular uma hipótese relativamente à origem da aleatoriedade exibida aquando da execução do programa?**

- ☐ Provavelmente o programa passa por `rand()`. Isso explicaria o resultado apresentado.
- ☐ O uso de `while()` é um forte candidato. O número de iterações é muito grande e isso pode estar a complicar a execução do programa.

- ☐ Aquela declaração na linha 24 onde `i = f1(--a)` é altamente duvidosa. Será que `a` é decrementado antes ou depois de ser passado como argumento?
- ☐ Creio que existem algumas hipóteses válidas mas o melhor mesmo é usar uma ferramenta de depuração para tirar todas as dúvidas!

### Tarefa 9 Task 9

Os procedimentos que se seguem requerem a instalação do editor de código Visual Studio Code. Pode usar esta [hiperligação](#) para descarregar esta aplicação ou, caso prefira, usar esta [hiperligação](#) para a instalar nos sistemas operativos Linux.

### Tarefa 10 Task 10

Instale a extensão C/C++, da Microsoft. Para aceder ao menu de extensões pode usar o comando `Ctrl+Shift+X`. De seguida, realize os seguintes passos:

1. No Visual Studio Code, abra a diretoria que contém o ficheiro com o código fonte;
2. Abra posteriormente o ficheiro `debug.c` no Visual Studio Code;
3. Pressione F5 ou `Run > Start Debugging`;
4. No topo da janela será exibido um menu *dropdown* para selecionar o ambiente. Selecione `C++ (GDB/LLDB)`;
5. No topo da janela será exibido um novo menu *dropdown* para selecionar uma configuração. Selecione `gcc`, sem nenhum número;

Se todo o processo correu sem problemas deverá ter uma nova pasta, denominada `.vscode`, com dois ficheiros `.json`, denominados `launch` e `tasks`.

Nota: caso tenha obtido algum erro, relativamente a `gdb`, instale o mesmo no seu Sistema Operativo. A título de exemplo, para Ubuntu, este poderá ser instalado via `sudo apt-get install gdb`.

### Tarefa 11 Task 11


Para avaliar potenciais problemas no programa serão usados pontos de paragem (*breakpoints*). Para tal basta clicar com o botão do lado esquerdo na linha onde deseja que o seu programa pare.

Coloque um ponto de paragem nas linhas 42 e 43 e pressione F5 (ou `Run > Start Debugging`). Nas tarefas e questões seguintes esta operação será identificada por **depuração**. De seguida, selecione o botão de depuração, no lado esquerdo do editor, ou use a combinação `Ctrl+Shift+D`. O programa irá parar no primeiro ponto de paragem.

Observe a secção de depuração. **Q13.: O que pode concluir em relação ao valor da variável `c` do programa?**


- ☐ Segundo a subsecção de Variables, `c` tem o valor de 3.
- ☐ Segundo a subsecção de Watch, não há variáveis no programa.
- ☐ Segundo a subsecção de Call Stack, está lá uma `main()` mas não sei que valor tem `c`.
- ☐ Nada. Existe uma subsecção de Breakpoints mas não me parece que o valor da variável `c` esteja lá...

### Tarefa 12 Task 12

Clique em F5 ou no primeiro botão , no topo da janela. O programa deverá prosseguir para o segundo ponto de paragem criado.

Observe, novamente, a secção de depuração. **Q14.: O que pode concluir relativamente à origem da aleatoriedade do resultado do programa?**

- ☐ Constato que a variável `c` tem, ainda, o valor 3. Logo a aleatoriedade deverá vir do `printf` ou `return 0`.
- ☐ Reparei, de forma muito perspicaz, que a variável `c` tem já um valor diferente e conclui, de forma ainda mais arisca, que a aleatoriedade deverá ter origem dentro da função `f`.

Termine o processo de depuração clicando no botão , no topo da janela, ou via `Shift+F5`.

### Tarefa 13 Task 13

Analise o ficheiro e registe as linhas onde colocaria pontos de paragem, visando a avaliação da origem da aleatoriedade do programa.

---

As tarefas e questões seguintes consideram os pontos de paragem nas linhas 13, 25, 36 e 37.

#### Tarefa 14 Task 14

Com o ficheiro `debug.c` selecionado, depure o programa e migre até ao segundo ponto de paragem (linha 13).

**Q15.: Tendo em conta a subsecção Call Stack, que funções foram chamadas?**

- ☐ Pela ordem apresentada, `main` chamou `f`, que chamou `f2`, e que por sua vez chamou `f1`.
- ☐ Aparentemente, `main` chamou `f2`.

**Q16.: Qual o valor de `a` e, dado este valor, qual a condição que será escolhida?**

- ☐ valor 3 e condição `if`    ☐ valor 3 e condição `else`
- ☐ valor 2 e condição `if`    ☐ valor 2 e condição `else`

#### Tarefa 15 Task 15

Avalie a sua resposta anterior utilizando a tecla F10 ou clicando em ↺, no topo da janela.

**Q17.: Conseguiu concluir algo relativamente à origem da aleatoriedade do resultado do programa?**

- ☐ Sim! Provém de `rand()`, dentro da função `f1`, como eu inicialmente tinha conjecturado!
- ☐ Ainda não consegui perceber de onde vem, mas sei que não provém do `rand()` da função `f1`.

#### Tarefa 16 Task 16

Prossiga para o próximo ponto de paragem (linha 25).

**Q18.: Dado o valor de `a` e `i`, qual será o valor de retorno da função `f2`?**

- ☐ 0    ☐ 1    ☐ 2    ☐ 3    ☐ 100000002    ☐ ∞

#### Tarefa 17 Task 17

Avalie a sua resposta anterior prosseguindo para o próximo ponto de paragem (linha 36).

**Q19.: De acordo com a subsecção Call Stack, em que função está, neste momento, o processo de depuração?**

- ☐ `fAlone`    ☐ `f2`    ☐ `f1`    ☐ `f`    ☐ `main`

**Q20.: Qual é a variável que propõe analisar para avaliar a sua resposta à questão 18?**

- ☐ `i`    ☐ `soma`    ☐ `array`    ☐ `tot`    ☐ `a`

#### Tarefa 18 Task 18

Clique em F5 (ou no botão ▶) até que chegue ao ponto de paragem na linha 37. A cada clique, tome atenção aos valores das variáveis `soma` e `i`.

**Q21.: Este ponto de paragem ajudou-o a concluir algo relativamente à origem da aleatoriedade do resultado do programa?**

- ☐ Creio que sim. O valor de `soma` tinha o somatório de todos os elementos do array mas quando saiu do ciclo ficou com um valor aleatório...
- ☐ Ainda não foi desta. Deduzo que o problema venha do próximo ponto de paragem, ou seja, no `printf`, da função `main`.

#### Tarefa 19 Task 19

Registe qual é, na sua opinião, o problema do programa que promove o aparecimento de uma resposta aleatória aquando da sua execução.

## 4 Perfilagem de Programas

### Program Profiling

A perfilagem (da expressão inglesa *profiling*) de um programa é uma análise dinâmica do programa que permite avaliar a percentagem de tempo total utilizado por cada função, que funções foram chamadas, qual a árvore de chamada das funções, entre outros aspetos. Neste guia laboratorial iremos usar o *profiler* `gprof`.

#### Tarefa 20 Task 20

Utilize o comando `gprof -v` para verificar se possui este *profiler* na sua máquina. Caso verifique que não se encontra instalado, deverá instalá-lo. Para Ubuntu, por exemplo, este poderá ser instalado da seguinte forma: `apt-get install binutils`.

Para obtermos informação de perfilagem do programa, este terá que ser habilitado aquando do processo de compilação. Para tal, terá que ser adicionada a opção `-pg` ao comando de compilação do programa.

#### Tarefa 21 Task 21

Compile o ficheiro `.c` disponibilizado com este guia fazendo uso da opção referida anteriormente.

**Q22.: Houve alguma alteração na diretoria onde o ficheiro foi compilado?**

- ☐ Houve, sim senhor. Foi criado um novo ficheiro com informação sobre o programa.
- ☐ Não creio. Parece-me que está tudo igual.

## Tarefa 22 Task 22

Execute o ficheiro resultante do comando usado para compilar na tarefa anterior.

**Q23.: E desta vez, verificou alguma alteração na diretoria onde o ficheiro foi executado?**

- ☐ Agora sim! Apareceu um ficheiro `pokemon.out`.
- ☐ Agora sim! Apareceu um ficheiro `.out`. É suposto executá-lo também?
- ☐ Não. E eu até fiz `ls -a` para ver se não estava algum ficheiro oculto!

Se realizou todos os passos com sucesso deverá ter um ficheiro denominado `gmon` na diretoria onde executou o programa. Para obter o conteúdo deste ficheiro e ser possível analisá-lo sugere-se a utilização do comando `gprof` que recebe como argumentos o executável do programa e `gmon.out`, por esta ordem.

**Q24.: Assumindo que o executável do programa tem o nome `a.out`, qual o comando que permite obter um ficheiro, denominado `profile_output.txt`, com o conteúdo resultante do comando `gprof`?**

- ☐ `gprof a.out gmon.out -o profile_output.txt`
- ☐ `gprof gmon.out a.out -o profile_output.txt`
- ☐ `gprof a.out gmon.out > profile_output.txt`
- ☐ `gprof gmon.out a.out > profile_output.txt`

No ficheiro criado na questão anterior poderá ver inúmeras características, devidamente descritas, resultantes da perfilagem do programa. Este ficheiro divide-se em duas partes: *Flat profile* e *Call graph*. No resto deste guia será analisada uma porção do *Flat profile*.

Atente à primeira tabela do *Flat profile*, com início na linha 3 do ficheiro `profile_output.txt`. **Q25.: Qual foi a função que, em termos percentuais, demorou mais tempo?**

- ☐ `fAlone`
- ☐ `f2`
- ☐ `f1`
- ☐ `f`
- ☐ `main`

**Q26.: Qual é a razão que encontra para que a função da resposta à questão anterior tenha demorado mais tempo?**

- ☐ A presença de `while` e `rand()` dentro da função é uma boa justificação.
- ☐ É uma função que tem um `while` a iterar um elevado número de vezes.
- ☐ A função tem um ciclo `for`.
- ☐ É a função principal e por isso tem direito a demorar o tempo que quiser.

**Q27.: Porque razão a função `fAlone` não aparece na tabela de *Flat profile*?**

- ☐ Porque não demorou praticamente tempo nenhum, logo não se justificava a sua presença.
- ☐ Porque esta função não foi, em momento algum, invocada ao longo do código. É código morto e deveria ser retirado do ficheiro fonte!

## Tarefa 23 Task 23

Modifique o ficheiro `Makefile` do guia laboratorial anterior para que, quando execute o comando `make profile`, o código fonte seja compilado com a opção `-pg`, executado e o conteúdo de `gmon.out` seja exportado para um ficheiro, denominado `profile_output.txt`. Esta exportação para o ficheiro `.txt` deverá usar o comando `gprof`.