

# Níveis de Abstração

## TPC4 e Guião laboratorial

Baseado no guia de Luís P. Santos, António J. Esteves e Alberto J. Proença

---

### Metodologia

Este documento é o **guião** de apoio ao funcionamento da primeira sessão laboratorial de SC. **É indispensável a sua leitura/estudo prévio**, incluindo a **entrega da ficha com a resolução dos exercícios propostos** no início da PL.

### Objetivo

Assimilar os vários **níveis de abstração** envolvidos no processo de desenvolvimento de *software* e as representações usadas em cada nível, bem como os **mecanismos de conversão** entre níveis.

Nesta sessão será usado um programa em C, constituído por 2 módulos (em ficheiros distintos). Os alunos deverão visualizar as várias fases do processo de conversão, **usando as ferramentas do Linux** `gcc`, `gdb` e `objdump`. Um texto de introdução ao Linux está disponível juntamente com este guião.

Para garantir resultados idênticos, deverão **usar sempre** a máquina virtual que se disponibiliza remotamente, em ambiente Linux, usando o **username** e **password** enviados por correio eletrónico na semana anterior à realização deste trabalho, pela equipa técnica do DI.

O **acesso remoto** faz-se com o protocolo `ssh` (*secure shell*). Em sistemas baseados em UNIX este cliente pode ser utilizado através das aplicações “Terminal”. Em sistemas Windows 10/11 pela PowerShell ou, em alternativa, através da aplicação `PuTTY`.

O comando a executar será o seguinte<sup>1</sup> (com o seu **username**), indicando depois a sua **password** :

```
ssh -c aes256-cbc -oKexAlgorithms=+diffie-hellman-group1-sha1 username@sc.di.uminho.pt
```

Uma vez na máquina remota escolha um editor de texto para começar a trabalhar. Recomendamos o `vi` (consultar <https://www.cs.colostate.edu/helpdocs/vi.html>) ou o `nano` (consultar, ignorando a parte da instalação, <https://phoenixnap.com/kb/use-nano-text-editor-commands-linux>).

---

## 1. Linguagem de alto nível (HLL)

Considere os módulos em C apresentados na tabela, que são para **colocar em 2 ficheiros separados**:

<b>prog.c</b>	<b>soma.c</b>
<pre>void soma (int); int main () {     int x=222;     soma (x);     return 1; }</pre>	<pre>int accum=0;  void soma (int p) {     accum += p; }</pre>

---

<sup>1</sup> Por se tratar de um sistema antigo, o cliente `ssh` deverá ser configurado para permitir o uso de cifras simétricas e de métodos de troca de chaves (*key exchange algorithms*, *KexAlgorithms*) entretanto descontinuados. Os algoritmos a utilizar podem ser especificados ao invocar o comando, nomeadamente a cifra `aes256-cbc` e a troca de chaves `diffie-hellman-group1-sha1`. Note-se que estas configurações deverão apenas ser utilizadas no contexto desta sessão.

**Exercício 1. (TPC)** Escolha um editor de texto e crie no servidor remoto os ficheiros **prog.c** e **soma.c**. Indique em que formato está representada a informação contida nestes ficheiros? Quantas células de memória são necessárias, nesse formato, para codificar e armazenar cada um dos caracteres do ficheiro?

**Exercício 2. (TPC)** Qual o tamanho de cada um dos ficheiros? Calcule os tamanhos manualmente e confirme os valores com um comando da *shell* do Linux (indique o comando que usar)

## 2. Compilação

Por **compilação** entende-se a conversão de um programa escrito num dado nível de abstração noutra de nível inferior. A maior parte dos utilitários atualmente conhecidos como “compiladores” permitem passar diretamente do nível HLL para o nível da linguagem máquina, efetuando na realidade 4 tarefas distintas: pré-processamento, compilação, montagem (recorrendo ao *assembler*) e ligação (recorrendo ao *linker*).

O manual das diversas versões do compilador C distribuído pelo projeto GNU, o *gcc*, estão disponíveis *online* em <http://www.gnu.org/software/gcc/onlinedocs/>, em que a versão disponível na máquina virtual é a 3.2.3. Um sumário muito compacto do manual duma versão do *gcc* é incluído no fim deste guião.

Compile o módulo **soma.c** usando o comando:

```
gcc -Wall -O2 -S soma.c
```

A opção `-Wall` ativa mensagens de diagnóstico para avisar que existem estruturas de código que são imprecisas ou para reportar erros, a opção `-O2` indica ao compilador para usar o nível dois de otimização do código, enquanto a opção `-S` indica que deve parar após gerar o código de montagem (*assembly*). Este comando gera o ficheiro **soma.s**.

**Exercício 3. (TPC)** Indique em que formato está representada a informação contida neste novo ficheiro?

**Exercício 4.** Usando um programa adequado visualize o conteúdo de **soma.s**. Além das mnemónicas das operações, dos nomes dos registos do IA-32 e das constantes numéricas, identifique outros tipos de informação textual existentes no ficheiro e qual o fim a que se destinam.

**Exercício 5. (TPC)** O programa especificado no ficheiro **soma.s** pode ser executado diretamente pela máquina? Em que nível de abstração se encontra?

## 3. Montagem (usando o *assembler*)

Utilize o comando:

```
gcc -Wall -O2 -c soma.s
```

para gerar o ficheiro **soma.o**, que contém código binário resultante da montagem do módulo **soma.s**. A opção `-c` indica que o processo termina após a montagem. O código binário não pode ser visualizado usando um editor de texto, pois o formato da informação já não é ASCII.

Para visualizar o conteúdo de um ficheiro objeto (binário) pode-se usar um **debugger** (depurador) fornecido com o Linux. Neste caso, para se iniciar o processo de depuração, far-se-ia:

```
gdb soma.o
```

Uma vez dentro do depurador, execute o comando:

```
(gdb) x/23xb soma
```

o qual irá examinar e mostrar (abreviado “**x**”) 23 “hex-formatted bytes” (abreviado para “**xb**”) a partir do início do código da função soma.

**Exercício 6.** O que representam os valores que está a visualizar? Explique o comportamento aparentemente anómalo deste último comando.

**Exercício 7. (TPC)** O programa guardado no ficheiro **soma.o** pode ser executado diretamente pela máquina? Em que nível de abstração se encontra? Obteria o mesmo resultado se substituísse o ficheiro **soma.s** pelo **soma.c** no comando *gcc*?

É possível ainda visualizar o código de montagem a partir do ficheiro objeto, quer dentro do depurador (com o comando **disass soma**), quer ainda usando um **disassembler** (desmontador) do Linux. Este tem a vantagem de mostrar ainda o código binário para além do código *assembly*.

Assim, abandone o `gdb` (com o comando **quit**) e execute o comando:

```
objdump -d soma.o
```

**Exercício 8.** O ficheiro **soma.o** desmontado deveria conter apenas linhas de código *assembly* da GNU com instruções do IA-32 e respetivos operandos. Contudo, encontra lá ainda outro tipo de informação. Identifique-o e encontre uma explicação para esse texto que lá encontra.

**Exercício 9.** (TPC) Como está representada a variável `accum`? Porque razão é ela representada desta forma?

**Exercício 10.** Quantas instruções tem a função `soma`? Quantos *bytes* ocupa cada instrução?

#### 4. Ligação (usando o *linker*) e execução

Para gerar o programa executável é necessário ligar os dois módulos entre si e com quaisquer outras funções de bibliotecas que sejam utilizadas, assim como acrescentar código de interface com o sistema operativo. Este é o papel do *linker*. Execute o comando:

```
gcc -Wall -O2 -o prog prog.c soma.o
```

**Exercício 11.** O resultado da execução deste comando é colocado no ficheiro `prog`. Usando o comando `file`, indique qual o formato da informação aí contida? Este ficheiro pode ser executado diretamente pela máquina?

Desmonte o programa gerado pelo comando anterior e guarde-o num ficheiro de texto, usando o comando:

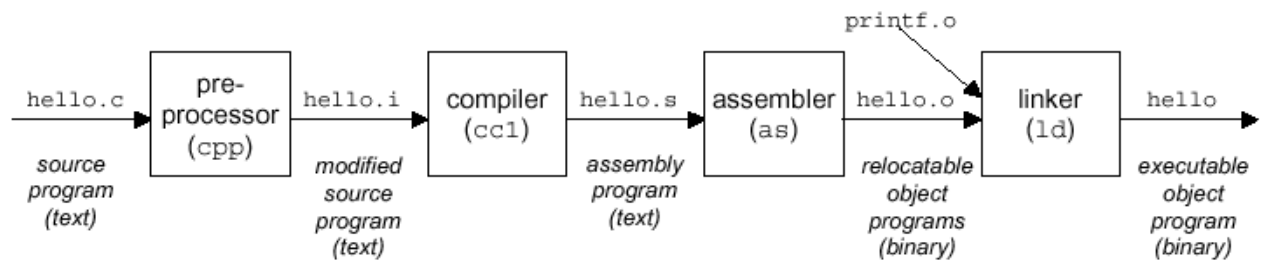
```
objdump -d prog > prog.dump
```

**Exercício 12.** (TPC) Localize a função `soma` no ficheiro `prog.dump`. Como está representada a variável `accum`?

**Exercício 13.** Consultando o ficheiro `prog.dump`, indique a ordem pela qual são armazenados em memória os 4 *bytes* correspondentes ao endereço de `accum`: *little-endian* ou *big-endian*? Justifique a sua resposta.

**Exercício 14.** Como é que a função `main` passa o controlo (invoca) para a função `soma`?

## 5. Excerto do manual de gcc



GCC(1)

GNU Tools

GCC(1)

### NAME

`gcc` - GNU project C and C++ Compiler (gcc-3.2.3)

### SYNOPSIS

```
gcc [-c|-S|-E] [-g] [other options] -O2 -o outFile inFile1 inFile2 ...
```

### DESCRIPTION

The `gcc` compiler processes input files through one or more of four stages: preprocessing, compilation, assembly, and linking. The “overall options” allow you to stop this process at an intermediate stage.

For any given input file, the file name suffix determines what kind of processing is done:

- `file.c` C source code which must be preprocessed, compiled, assembled.
- `file.i` C source code which should not be preprocessed, only compiled and assembled.
- `file.s` Assembler code that must be assembled.
- `file.o` An object file to be passed to the linker. File names with no recognized suffix are treated this way.

### OPTIONS

#### Common Options

- c** Compile or assemble the source files, but do not link.
- S** Stop after the stage of compilation proper; do not assemble.
- E** Stop after the preprocessing stage.
- g** Produce debugging information that can be used by `gdb`.
- O2** `gcc` performs nearly all supported optimizations that avoids a space-speed tradeoff.
- o file** Place output in file *file*; if `-o` is not specified, the default is to put in `a.out`.

#### C Language Options [...]

#### Warning Options [...]

#### Debugging Options [...]

#### Optimization Options [...]

#### Preprocessor Options [...]

#### Assembler Option [...]

#### Linker Options [...]

#### Machine Dependent Options [...]

**Nº****Nome:****Turma:**

### Resolução dos exercícios

**Nota:** Apresente os cálculos que efetuar; o não cumprimento desta regra equivale à não entrega do trabalho.

1. **Crie** os ficheiros `prog.c` e `soma.c` no servidor remoto.
2. Qual o tamanho de cada um dos ficheiros `prog.c` e `soma.c`? Calcule os tamanhos manualmente **e** confirme os valores com um comando da *shell* do Linux.
3. O ficheiro `soma.s` pode ser executado diretamente pela máquina? Em que nível de abstração se encontra?
5. **Indique (i)** se o programa no ficheiro `soma.s` pode ser executado diretamente pela máquina (justifique a resposta) e **(ii)** em que nível de abstração se encontra.
7. **Indique (i)** se o programa no ficheiro `soma.o` pode ser executado diretamente pela máquina (justifique a resposta) e **(ii)** em que nível de abstração se encontra.
9. Como está representada a variável `accum` no ficheiro `soma.o`? Porque razão é ela representada desta forma?
12. Consultando o ficheiro `prog.dump`, resultante de desmontar o programa executável `prog`, como está representada a variável `accum`?