

# Níveis de Abstração

## TPC4 e Guião laboratorial

Luís Paulo Santos e Alberto José Proença

---

### Objetivo geral

Este documento é o **guião** para apoio ao funcionamento da primeira sessão laboratorial de SC. **É indispensável a sua leitura/estudo prévio**, incluindo a resolução e **entrega dos exercícios propostos**.

### Objetivo concreto

Assimilar, ao longo de uma sessão laboratorial, os vários **níveis de abstração** envolvidos no processo de desenvolvimento de *software* e respectivas representações usadas em cada nível, bem como os **mecanismos de conversão** entre esses níveis.

Para atingir estes objetivos: desenvolver um pequeno programa em C, constituído por 2 módulos, e acompanhar e visualizar as várias fases, **usando ferramentas do Linux**, `gcc`, `gdb` e `objdump`.

Para garantir idênticos resultados em todos os trabalhos, **usar sempre** a máquina virtual que se disponibiliza remotamente. O acesso faz-se executando "`ssh lei-1011@sc.di.uminho.pt`" numa consola Unix (ver a *password* na plataforma *e-learning*).

---

## 1. Linguagem de alto nível (HLL)

Escreva em 2 ficheiros, usando o editor de texto que considerar mais adequado, os 2 módulos em C apresentados na tabela.

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

**Exercício 1.** (TPC) Em que formato está representada a informação contida nestes ficheiros?

**Exercício 2.** (TPC) Qual o tamanho de cada um dos ficheiros?  
(Calcule manualmente e verifique com um comando da *shell* do Linux)

## 2. Compilação

Por **compilação** entende-se a conversão de um programa escrito num dado nível de abstração noutra de nível inferior. Historicamente o termo surgiu da conversão de um programa escrito numa HLL para o nível do *assembly*. Contudo, a maior parte dos utilitários actuais conhecidos como “compiladores” permitem, com uma única linha de comando, passar diretamente do nível HLL para o nível da linguagem máquina, executando na realidade 4 programas distintos, correspondentes a 4 fases diferentes: pré-processamento, compilação, montagem (com o *assembler*) e união (com o *linker*). Uma descrição mais detalhada destas fases encontra-se no texto que acompanha as aulas teóricas (*Introdução aos Sistemas de Computação*, Cap.3, com material retirado do livro CSAPP).

As diversas versões do manual do compilador de C distribuído pelo projecto 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
```

O *switch* `wall` activa o envio de mensagens de diagnóstico relatando a existência de estruturas de código que poderão conter imprecisões ou potenciais fontes de erro, o *switch* `O2` indica ao compilador para usar o nível dois de otimização do código, enquanto o *switch* `S` indica que deve gerar apenas o código *assembly*. Este comando gera o ficheiro `soma.s`.

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

**Exercício 4.** Usando um programa adequado visualize o conteúdo de `soma.s`. Encontra informação simbólica neste programa? Qual?

**Exercício 5.** (TPC) Este programa (o ficheiro `soma.s`) pode ser executado diretamente pela máquina? Em que nível de abstração se encontra?

### 3. Compilação e montagem (uso do *assembler*)

Use o comando

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

para gerar o ficheiro `soma.o` (código binário resultante de compilação e montagem do módulo `soma.c`); o *switch* `-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 objecto (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, pode-se activar 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?

**Exercício 7.** (TPC) Este programa (o ficheiro `soma.o`) pode ser executado directamente pela máquina? Em que nível de abstração se encontra?

O conteúdo dum ficheiro objecto pode também ser visualizado usando ***disassemblers*** (desmontadores), com uma vantagem: estes geram também código *assembly* a partir do ficheiro objecto. Execute o comando

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

**Exercício 8.** Este programa contém informação simbólica? Qual?

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

**Exercício 10.** Quantas instruções tem a função `soma`? Quantos *bytes* ocupa? Quais são as instruções mais curtas e quais as mais longas?

#### 4. União (uso do *linker*) e execução

Para gerar o programa executável é necessário ligar os dois módulos entre si e com quaisquer outras bibliotecas de funções que sejam utilizadas, assim como acrescentar código que faz o 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`. Qual o formato da informação aí contida? Este ficheiro pode ser executado directamente pela máquina?

Desmonte este programa executável e guarde-o num ficheiro de texto, usando o comando

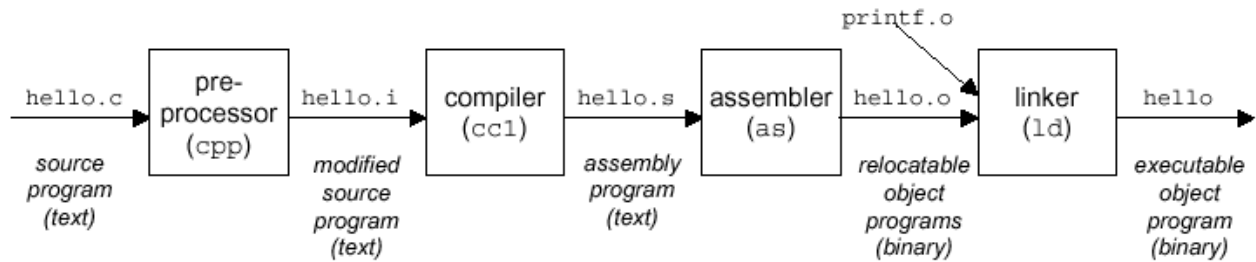
```
objdump -d prog > prog.dump
```

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

**Exercício 13.** Porque ordem são armazenados na memória os 4 *bytes* correspondentes ao endereço de `accum`? *Little-endian* ou *big-endian*?

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

## 5. Sumário do manual de gcc



GCC(1)

GNU Tools

GCC(1)

### NAME

gcc, g++ - GNU project C and C++ Compiler (gcc-3.43)

### SYNOPSIS

gcc [ option | filename ]...

### DESCRIPTION

The C and C++ compiler are integrated. Both process input files through one or more of four stages: preprocessing, compilation, assembly, and linking. Source file-name suffixes identify the source language, but which name you use for the compiler governs default assumptions:

gcc assumes preprocessed (.i) files are C and assumes C style linking.

Suffixes of source file names indicate the language and kind of processing to be done:

- .c C source; preprocess, compile, assemble
- .i preprocessed C; compile, assemble
- .s Assembler source; assemble
- .o Object file: pass to the linker.

### OPTIONS

#### Overall Options

-c -S -E -o file -pipe -v -x language

#### C Language Options

#### Warning Options

#### Debugging Options

#### Optimization Options

#### Preprocessor Options

#### Assembler Option

#### Linker Options

#### Machine Dependent Options

<b>Nº</b>	<b>Nome:</b>	<b>Turma:</b>
-----------	--------------	---------------

### Resolução dos exercícios

(Nota: Apresente sempre os cálculos que efectuar no verso da folha; o não cumprimento desta regra equivale à não entrega do trabalho.)

1. **Indique** em que formato está representada a informação contida nos ficheiros `prog.c` e `soma.c`.
2. **Indique** qual o tamanho da cada um dos ficheiros. Calcule manualmente e verifique com um comando da *shell* do Linux (que comando usou?).
3. **Indique** em que formato está representada a informação contida no ficheiro `soma.s`.
5. **Indique** (i) se o programa no ficheiro `soma.s` pode ser executado directamente pela máquina, e (ii) em que nível de abstração se encontra.
7. **Indique** (i) se o programa no ficheiro `soma.o` pode ser executado directamente pela máquina, e (ii) em que nível de abstração se encontra.