Níveis de Abstracção

Trabalho para Casa: **TPC3**

Luís Paulo Santos e Alberto José Proença

Objectivo geral

Ler atentamente este documento, o qual servirá de **guião** para o funcionamento da sessão laboratorial da semana de 21-Out-02.

Objectivo concreto

Assimilar, ao longo de uma aula prática, os vários **níveis de abstracção** 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 objectivos ir-se-á desenvolver um pequeno programa em C, constituído por 2 módulos, e, **usando ferramentas do Linux** - `gcc`, `gdb` e `objdump` - acompanhar e visualizar as várias fases desse processo.

---

1. Linguagem de alto nível (*HLL*)

Escreva em C, usando o editor de texto que considerar mais adequado, os 2 módulos apresentados na tabela 1.

prog.c	**soma.c**

`main ()`
`{`
`  int x;`
 	
`  soma (x);`
`}`
	`int accum=0;`
	
	`void soma (int p)`
	`{`
	`  accum += p;`
	`}`

Questão 1 – Qual o tamanho da cada um dos ficheiros?

Questão 2 – Em que formato está representada a informação contida nestes ficheiros?

2. Compilação

Por **compilação** entende-se a conversão de um programa escrito num dado nível de abstracção noutro 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 directamente do nível HLL para o nível da linguagem máquina, executando na realidade 4 programas distintos, correspondentes a 4 fases diferentes: pré-processamento, compilação, montagem (*assembler*) e *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). Um sumário muito compacto do manual do `gcc` é incluído no fim deste guião.

Compile o módulo `soma.c` usando o comando

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

O *switch* O2 indica ao compilador para usar o nível dois de optimização do código, enquanto o *switch* S indica que deve gerar apenas o código *assembly*. Este comando gera o ficheiro `soma.s`.

Questão 3 – Em que formato está representada a informação contida neste ficheiro?

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

Questão 5 – Este programa pode ser executado directamente pela máquina? Em que nível de abstracção nos encontramos?

3. Montagem (*Assembler*)

Use o comando

```
gcc –O2 –c soma.c
```

para gerar o código binário correspondente ao módulo `soma.c`. 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, far-se-ia:

```
gdb soma.o
```

Uma vez dentro do depurador, pode-se enviar o comando:

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

o qual irá e**x**aminar e mostrar (abreviado "**x**") 23 "he**x**-formatted **b**ytes" (abreviado para "**xb**") do ficheiro `soma`.

Questão 6 – O que representam os valores que está a visualizar?

Questão 7 – Este programa pode ser executado directamente pela máquina? Em que nível de abstracção nos encontramos?

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

```
objdump –d soma.o
```

Questão 8 – Este programa contem informação simbólica?

Questão 9 – Como está representada a variável `accum`? Porque razão é ela representada desta forma?

Questão 10 – Quantas instruções tem a função soma? Quantos *bytes* ocupa? Quais são as instruções mais curtas e mais longas?

4. *Linker*

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 lida com o Sistema Operativo. Este é o papel de *linker*. Execute o comando

```
gcc –O2 –o prog prog.c soma.o
```

Questão 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?

Visualize o conteúdo deste ficheiro e guarde-o num ficheiro de texto usando o comando

```
objdump –d prog > prog.dump
```

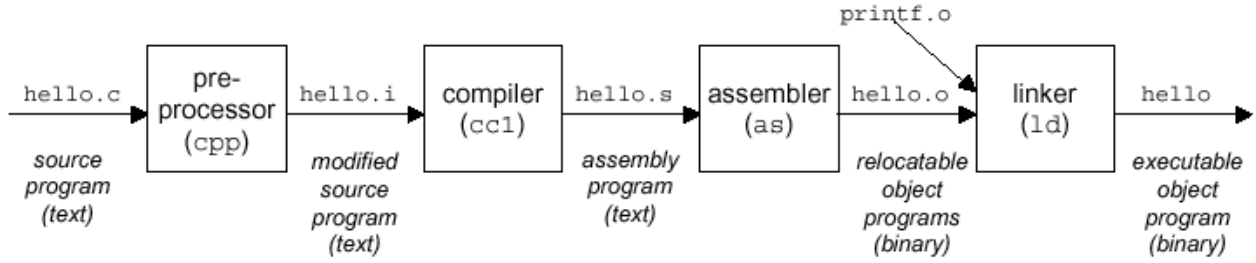
Localize no ficheiro `prog.dump` a função `soma`.

Questão 12 – Como está representada a variável `accum`?

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

Questão 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-2.96)

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

Language Options

Warning Options

Debugging Options

Optimization Options

Preprocessor Options

Assembler Option

Linker Options

Configuration Dependent Options

Nº	**Nome:**	**Turma:** Seg1 - Seg2 - Ter

Resposta às questões

Q1.

Q2.

Q3.

Q4.

Q5.

Q6.

Q7.

Q8.

Q9.

Q10.

Q11.

Q12.

Q13.

Q14.