FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Engenharia Electrotécnica

PROGRAMAÇÃO DE MICROPROCESSADORES

2021 / 2022

Mestrado Integrado em Engenharia Electrotécnica
e de Computadores

1º ano
1º semestre

Trabalho nº 9
Ficheiros

(este trabalho ***não corresponde*** a nenhuma aula de laboratório**)**

http://tele1.dee.fct.unl.pt/pm

Paulo da Fonseca Pinto
Luis Bernardo

1 Introdução

No capítulo 10 do livro "Linguagem C" de Luís Damas, recomendado para a disciplina de Programação de Microprocessadores, são apresentados os ficheiros. Esta aula visa consolidar estas matérias através de um conjunto de exercícios. Faça todos os exercícios pedidos em ficheiros separados e **GUARDE O CÓDIGO desenvolvido na memória USB**. Durante a aula o docente pode pedir-lhe para mostrar o código desenvolvido.

2 Dois tipos de ficheiros

Existem dois tipos de ficheiros: ficheiros em **modo de texto** e em **modo binário**.

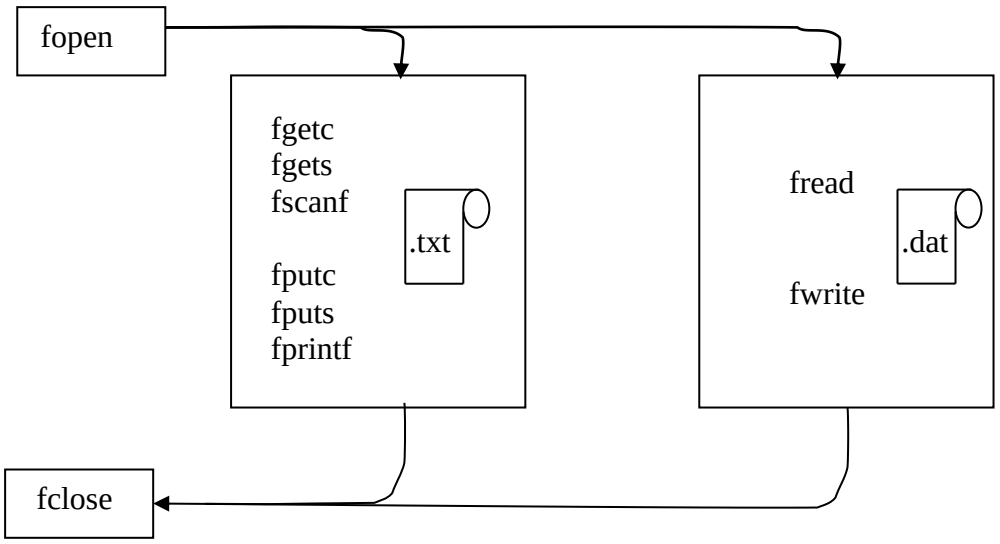
- Um ficheiro em modo de texto é constituído por caracteres que são perceptíveis por nós, humanos (alfabeto e os acrescentos tradicionais).
- Os ficheiros binários são constituídos por qualquer conjunto de bits e o significado só é perceptível pelo programa que o escreve e pelo que o lê.

Nos ficheiros de texto existe ainda o problema de como as linhas acabam:

- Em Linux existe apenas o carácter NewLine '\n'
- No mundo Windows existem dois: o Carriage Return '\r' e o NewLine '\n'
- No mundo MacOS existe apenas o Carriage Return '\r'

Bonito, hein?

A figura em baixo mostra as diferenças e semelhanças dos dois tipos de ficheiros. Basicamente os dois tipos de ficheiros abrem-se e fecham-se com instruções muito parecidas mas as operações de escrita e leitura são totalmente diferentes. Na figura estão mostradas apenas as funções mais usadas de leitura e escrita. As terminações usadas para os ficheiros (.txt e .dat) são apenas ilustrativas para a figura. Não existe obrigatoriedade alguma de o nome do ficheiro ter uma terminação que indique o seu modo.



3 Dois ficheiros já construídos

Para este trabalho existem dois ficheiros já construídos com 20 antigos (esperemos) alunos de PM. Os ficheiros têm os nomes de `cobaia.txt` e `cobaia.dat`. Um dos ficheiros está em modo de texto e outro em modo binário. O ficheiro binário contém sequências da estrutura definida no trabalho da aula anterior. Como só existem inteiros e vectores de caracteres, é trivial.

Ao fazer o código para ler um e outro vai perceber as vantagens e inconvenientes de usar um dos modos e o outro. Não existem realmente vantagens absolutas de um relativamente a outro, mas algumas desvantagens são aborrecidas.

> **ATENÇÃO:** A maior parte dos editores de texto (crimson, gedit, etc.) já compensam o facto de haver um carácter ou dois no final de uma linha nos ficheiros de texto.
>
> Assim, nem nos apercebemos se os ficheiros que estamos a abrir foram feitos no mundo Windows, Linux ou MacOS.
>
> Com os ficheiros deste trabalho as coisas não são bem iguais porque vai ser o nosso programa a abri-los e não os editores de texto. Os ficheiros foram feitos no mundo Linux.

4 Uso de ficheiros

4.1 Exercício 1: O programa

Use o programa da aula anterior acrescentando funções de leitura e escrita em ficheiros binários e de texto.

O menu principal é o mostrado em baixo em que a parte a vermelho pertence à aula anterior.

/********** PROGRAMA SIRP – Registo dos cidadaos **********/

a – Mostrar os número de série de todos os elementos da base de dados
b – Mostrar toda a informação do cidadão com um certo número (a pedir)
c – Modificar a informação do cidadão com um certo número (a pedir)
d – Criar um novo registo para um novo cidadão
e – Apagar o registo com um certo número (a pedir)
f – Ler a base de dados de um ficheiro em modo binário (nome a pedir)
g – Ler a base de dados de um ficheiro em modo de texto (nome a pedir)
h – Escrever a base de dados num ficheiro em modo binário (nome a pedir)
i – Escrever a base de dados num ficheiro em modo de text (nome a pedir)

s – Sair do programa

OBSERVAÇÂO: Um verdadeiro programador pensará que um ficheiro de texto ou o *stdin* ou o *stdout* são coisas muito parecidas e que se pode poupar imenso trabalho!...

Um programador nabo vai repetir código desnecessariamente.

4.2 EXERCÍCIO 2: TRÊS MODOS DE CORRER O PROGRAMA

Pretende-se agora que o programa arranque de um modo mais automático, fazendo logo a leitura de um ficheiro e só depois se comporte como no exercício 1 mostrando o menu. Assim, assumindo que o programa se chama de `base`, o modo normal de o correr do exercício 1 é:

Modo Normal:

```
$ base
```

Neste caso o programa deve entrar logo para o menu principal.

O segundo modo de o correr é lendo directamente o ficheiro de texto que é fornecido na linha de comando:

Modo Texto:

```
$ base fich.txt
```

Neste caso o programa deve ler o ficheiro chamado "fich.txt" e carregar a informação na sua base de dados. Depois deve entrar para o menu principal.

O terceiro modo de o correr é lendo directamente o ficheiro binário que é fornecido na linha de comando:

Modo Binário:

```
$ base fich.dat
```

Neste caso o programa deve ler o ficheiro binário chamado "fich.dat" e carregar a informação na sua base de dados. Depois deve entrar para o menu principal.

4.3 EXERCÍCIO 3: AINDA MAIS UM MODO PARA CORRER O PROGRAMA

A partir do ficheiro `cobaia.txt` crie um novo ficheiro de texto chamado `cobaia2.txt` com apenas cinco cidadãos.

Edite esse ficheiro e coloque-lhe mais alguma informação que sinta que vai ser necessária, nomeadamente as escolhas dos vários menus que vão sendo propostos ao utilizador, mas que agora tem de ser o ficheiro `cobaia2.txt` a fornecer.

O objetivo é que o programa possa correr com o seguinte comando de linha.

```
$ base < cobaia2.txt
```

O que se pretende é que o programa escreva a base de dados (os cinco cidadãos) num ficheiro binário chamado `resultado.dat` e que depois termine.

4.4 EXERCÍCIO 4: PROCESSAMENTOS ADICIONAIS

Este trabalho pode ser muito rápido para os alunos que já estejam muito à vontade em programação. Para esses alunos é sugerido fazerem outros procedimentos como, por exemplo:

Tornar o mundo verde: Ler a base de dados e modificar o clube de todos os cidadãos para o Sporting Clube de Portugal. Escrever a base de dados noutro ficheiro.

Isolar cidadãos: Escrever um ficheiro com todos os cidadãos com uma característica (por exemplo, olhos azuis, altura acima de um certo valor, etc.).

4.5 EXERCÍCIO 5: PARA OS VERDADEIROS PROFISSIONAIS – OS PRÓ

Para os verdadeiros profissionais sugere-se a alteração do programa para que não exista limite para a base de dados.

Uma possibilidade é o ficheiro ter um primeiro campo a indicar quantos cidadãos estão lá contidos, mas o verdadeiro desafio é nem haver isso – o ficheiro vai sendo lido até acabar. Assuma ainda um número máximo de cidadãos para não ter de usar estruturas dinâmicas (ver abaixo).

4.6 EXERCÍCIO 6: PARA OS TÃO PROFISSIONAIS QUE AINDA NEM EXISTE NOME PARA ELES

O "máximo" que se pode pensar neste trabalho será efectuar as seguintes duas transformações:

1. Use estruturas dinâmicas em que cada cidadão seja um bloco de memória e se vá pedindo memória à medida que existam cidadãos. Para ligar estes blocos de memória e os poder percorrer use listas duplamente ligadas. Tem de escrever funções para poder retirar e acrescentar blocos, por exemplo.
2. Divida o código por mais do que um ficheiro.

Para a segunda transformação a compilação do código e a produção do ficheiro executável pode dar trabalho. A ferramenta ***make*** é muito útil nestes casos. A secção seguinte, retirada de um trabalho de anos anteriores, explica como usar a ferramenta ***make*** para dois ficheiros de codigo ("**.c**") e um ficheiro de cabeçalhos ("**.h**").

5 Makefile

O uso da ferramenta ***make*** começa a ser interessante quando o programa começa a ter muitos ficheiros. Imagine um programa com 30 ficheiros de código (".c") e uns 10 de *include* (".h"). Imagine que mudava o código num dos ficheiros de código. Não faz sentido compilar todos os ficheiros de código, mas apenas o que mudou. Assim, deve-se agora usar a opção "-c" no compilador para ele produzir apenas os ficheiros objeto (".o") de cada ficheiro de código e uma instrução final para *linkar* todos os ficheiros objeto para produzir o ficheiro executável.

Este nosso trabalho ainda é muito pequenino, mas o ficheiro **Makefile** apresentado a seguir tem a seguinte interpretação. O objetivo principal é produzir o executável *lista_alunos*. Outro objetivo que é tratado se se executar "*make clean*" é apagar todos os ficheiros objeto, todos os ficheiros terminados com "~", assim como o executável.

Para o objetivo principal é dito que o executável depende dos tempos de modificação dos dois ficheiros objeto indicados. Caso algum deles tenha um tempo de modificação superior ao tempo do ficheiro executável, eles vão ter de ser atualizados e depois executa-se a operação de *linkagem* ("*gcc –o*"). Para os atualizar vai-se seguir o que é dito na linha respetiva que tem a mesma estrutura (de quem depende e o que se deve fazer).

Também pode pensar na **Makefile** ao contrário, a começar por baixo. Isto é, para cada ficheiro objeto é visto se ele está atualizado referentemente aos ficheiros de que depende, e depois é visto o executável referentemente aos ficheiros objeto.

```
all: lista_alunos

lista_alunos: lista_alunos.o aluno.o
  gcc -o lista_alunos lista_alunos.o aluno.o

lista_alunos.o: lista_alunos.c aluno.h
  gcc -c lista_alunos.c

aluno.o: aluno.c aluno.h
  gcc -c aluno.c

clean:
  rm -f *~ *.o lista_alunos
```

Existe um comando do Linux, designado por *touch*, que "toca" no ficheiro, atualizando-lhe a data de modificação. Experimente fazer

```
$ touch aluno.c
```

ou

```
$ touch aluno.h
```

e depois fazer *make*. Vai ver a ferramenta *make* executar o que deve para atualizar o ficheiro executável.