DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

PROGRAMAÇÃO

(LEEC 24/25)

Enunciado do Projeto

Ortografia

1. Introdução

O trabalho que se descreve corresponde ao projeto da UC de Programação para o curso LEEC em 2024/2025. O projeto a realizar corresponde ao desenvolvimento de um programa que implemente mecanismos para verificar a ortografia de um texto.

O objetivo do projeto é promover e avaliar a prática dos conceitos e técnicas de programação estudados no âmbito desta UC. O projeto deverá ser realizado ao longo do período, acompanhando a matéria teórica lecionada nesta UC. Conforme descrito na secção de avaliação, haverá uma entrega intermédia, e uma entrega final que corresponde à entrega do projeto completo.

2. Descrição do Problema a Resolver

O programa deve poder funcionar com dados lidos do teclado, e o resultado do programa pode ser escrito no écran. Alternativamente, o programa deve poder funcionar com ficheiros, podendo ser parametrizado através de argumentos na linha de comando. De seguida descrevem-se as várias funcionalidades a implementar, em conjunto com exemplos.

2.1. Dicionário

O programa deve começar por ler um dicionário de um ficheiro para a memória, para uma consulta mais rápida. O ficheiro pode corresponder a um dicionário de Inglês, um dicionário de Português, ou de uma outra língua. Dois exemplos de ficheiros de dicionário que podem ser utilizados são:

`/usr/share/dict/words`	dicionário de Inglês Americano com 104334 palavras;
`/usr/share/dict/portuguese`	dicionário de Português com 431102 palavras.

Os dois ficheiros encontram-se disponíveis, nos locais indicados, nos computadores dos laboratórios da SCDEEC. Os mesmos foram também disponibilizados na página da disciplina.

Poderão também ser utilizados outros ficheiros de dicionário mais pequenos, e.g. para testes durante o desenvolvimento do programa, pois o tempo de execução com ficheiros de dicionário grandes pode ser significativo.

São adoptadas as seguintes regras ao ler o dicionário de um ficheiro:

- cada linha do ficheiro só tem uma única palavra;
- a palavra termina quando aparecer espaço, tab, mudança de linha, ou o caracter '/'

Estas regras permitem utilizar os dicionários da aplicação "spell" e os dicionários "hunspell", amplamente usados em sistemas UNIX.

Não se sabendo a dimensão do ficheiro, ele deve ser lido para um vector alocado dinamicamente. No entanto, para testes iniciais (e para a entrega intercalar da fase 1), poderá ser usado um vector estático, sendo feita a leitura de um número definido *a priori* de palavras do dicionário.

Após ler o ficheiro de dicionário para memória, o vector de palavras deve ser ordenado alfabeticamente, ignorando minúsculas e maiúsculas. Sugere-se usar a função qsort() com a função de comparação strcasecmp(). A ordenação alfabética do dicionário permite posteriormente usar pesquisa binária no vector do dicionário com a função bsearch(), que é muito mais rápida que uma pesquisa sequencial no dicionário.

2.2. Modo de funcionamento 1: detecção de palavras erradas

No modo de funcionamento 1, o programa deve ler o texto de entrada linha a linha e escrever na saída as linhas com palavras erradas, seguidas de uma linha por cada palavra errada detectada nessa linha. Considera-se uma palavra errada se não constar no dicionário lido.

As linhas de saída a escrever são:

- a linha da entrada com o erro, precedida do número da linha, um caracter ':' e um espaço;
- uma linha com a mensagem "Erro na palavra ", em conjunto com a palavra errada dentro de aspas, por cada palavra errada nessa linha.

Como exemplo, considere a seguinte linha de entrada:

```
"You may address me as the Count Von Kramm, a Bohemian nobleman.
```

Como há duas palavras que não constam do dicionário nessa linha, o programa escreveria a linha apenas uma vez, seguida de duas linhas com as palavras erradas:

```
1: "You may address me as the Count Von Kramm, a Bohemian nobleman.
Erro na palavra "Von"
Erro na palavra "Kramm"
```

Ao ler o texto de entrada, considera-se que uma palavra pode ser composta de letras minúsculas, maiúsculas, plica (desde que não seja o primeiro caracter da palavra nem o último caracter da palavra) e ainda caracteres da string "àáéíóúãõâêôçÀÁÉÍÓÚÃÕÂÊÔÇ", para aceitar palavras portuguesas.

Desta forma, temos exemplos de palavras válidas como:

```
o'clock                    man's                    barões
```

Texto de entrada que não forme palavras deve ser ignorado, tal como pontuação, plicas (que não estejam no meio de palavras), aspas, e outros símbolos.

Na verificação se cada palavra está no dicionário, deve-se ignorar a "capitalização" das letras, ou seja, deve-se aceitar palavras com minúsculas, com maiúsculas, ou com mistura de minúsculas e maiúsculas. Pode usar a função de comparação strcasecmp(). Caracteres portugueses acentuados não vão funcionar com esta função, mas não precisam de ser preocupar com esta questão.

2.3. Modo de funcionamento 2: sugestões de palavras alternativas

No modo de funcionamento 2, o programa deve ler o texto de entrada linha a linha e escrever as mesmas linhas que no modo de funcionamento 1 quando detecta palavras erradas, mas apresentando uma linha adicional a cada palavra errada com alternativas, tiradas do dicionário, até um número máximo de alternativas especificado na execução do programa. As alternativas devem ser separadas por uma vírgula e um espaço. No caso de não encontrar alternativas no dicionário, nas condições especificadas, o programa deve escrever uma linha em branco.

As alternativas devem ser apresentadas pela ordem de menor número de diferenças em primeiro lugar, e pela ordem do dicionário em segundo lugar. O número máximo de diferenças é especificado na execução do programa. Considera-se uma diferença ter uma letra trocada, uma letra a mais no início ou fim da palavra, uma letra a menos no início ou fim da palavra, uma letra a menos ou a mais

no meio da palavra. Considera-se ainda uma diferença partir uma palavra em duas palavras diferentes, que existam no dicionário.

Considere-se, como exemplo, a palavra “centred” que não existe no dicionário. As alternativas do dicionário apresentadas, com um máximo de duas diferenças, são, por ordem:

```
Erro na palavra "centred"
cent red, centered, central, enured, gentled, tenured
```

As duas primeiras alternativas têm uma diferença, enquanto as restantes têm duas diferenças.

O algoritmo para obter as diferenças deve ser o seguinte:

Primeiro percorre-se o dicionário todo à procura de 1 diferença, depois percorre-se o dicionário todo à procura de 2 diferenças, e assim sucessivamente até ao número máximo de diferenças especificado.

Por cada palavra do dicionário, percorrem-se as letras da palavra errada a partir do início.

Na primeira letra diferente, define-se *offset* como o número de diferenças que se está à procura, para saltar *offset* caracteres na palavra errada ou na palavra do dicionário a partir do primeiro caracter diferente.

Se chegar ao fim da palavra do dicionário e não ao fim da palavra errada, e ainda não encontrou diferenças e está à procura de apenas 1 diferença, verifica se o resto da palavra errada está no dicionário: neste caso, achou uma alternativa que é partir a palavra errada em duas palavras do dicionário. É o caso em que se obtém “cent red” para a palavra errada “centred”. Embora todos os caracteres sejam iguais, considera-se 1 diferença.

Contabiliza-se 1 diferença por cada caracter diferente a partir do início das palavras, sem considerar o offset. Isto permite encontrar “central” com 2 diferenças em relação à palavra errada “centred”. Também permite encontrar “tenured” com 2 diferenças em relação à palavra errada “centred”.

Contabiliza-se 1 diferença por cada caracter diferente a partir do início das palavras considerando o *offset* na palavra do dicionário. Isto permite encontrar “centered”, que é igual à palavra errada “centred” nos primeiros 4 caracteres, tem a letra ‘c’, sobre a qual se salta com *offset* 1, tendo o resto da palavra igual, resultando numa diferença de 1 para “centred”.

Contabiliza-se 1 diferença por cada caracter diferente a partir do início das palavras considerando o *offset* na palavra errada. Isto permite com a palavra errada “centetered” encontrar “centered”, saltando um *offset* de 2 caracteres a partir do caracter 5, que é o primeiro diferente, resultando numa diferença de 2 caracteres.

Se chegar ao fim da palavra do dicionário, contabiliza tantas diferenças adicionais quantos os caracteres que sobraram na palavra errada. Isto permite encontrar “cent” com 3 diferenças em relação à palavra errada “centred”.

Se chegar ao fim da palavra errada, contabiliza tantas diferenças adicionais quantos os caracteres que sobraram na palavra do dicionário. Isto permite encontrar “centrist” com 3 diferenças em relação à palavra errada “centred”.

Por cada palavra do dicionário, percorrem-se as letras da palavra errada a partir do fim para o início.

Contabiliza-se 1 diferença por cada caracter diferente a partir do fim das palavras. Isto permite encontrar "gentled" com 2 diferenças em relação à palavra errada "centred". Neste caso, procurar a partir do fim ou do início, permite obter o mesmo.

Se chegar ao início da palavra do dicionário, contabiliza tantas diferenças adicionais quantos os caracteres que sobraram na palavra errada. Isto permite encontrar "enured" com 2 diferenças em relação à palavra errada "centred". Neste caso, procurando a partir do início, todos caracteres seriam diferentes, resultando em 7 diferenças.

Se chegar ao início da palavra errada, contabiliza tantas diferenças adicionais quantos os caracteres que sobraram na palavra do dicionário. Isto permite com a palavra errada "ntered" encontrar a palavra de dicionário "centered" com 2 diferenças.

Encontrou uma palavra alternativa apenas se o número de diferenças encontrada é igual ao número de diferenças que andava à procura.

Em todas as comparações ignora-se a "capitalização" dos caracteres, muito embora a "capitalização" das palavras do dicionário encontradas deva ser mantida e apresentada tal como está no dicionário. Por exemplo, para a palavra errada "Papier", deve ser apresentado, assumindo que se apresentam no máximo 10 alternativas:

```
Erro na palavra "Papier"
Napier, PA pier, panier, paper, rapier, apter, babier, cagier, copier,
dapper
```

Nas palavras alternativas apresentadas não pode haver palavras duplicadas. No caso de uma palavra errada poder ser transformada numa palavra do dicionário com um número variável de modificações, só a que tem menos modificações será apresentada. Considere-se, como exemplo, a palavra "atualization" que não consta do dicionário, havendo no dicionário a palavra "actualization". Há uma diferença ao se considerar apagar a letra 'c' de "actualization", mas também se pode considerar que existem duas diferenças caso se troque a letra 'c' por 'a' e se apague o 'a' inicial de "actualization", visto que "actualization" tem mais um caracter que "atualization". Neste caso, a palavra "actualization" só apareceria uma única vez na lista de alternativas na posição correspondente a ter apenas uma única diferença:

```
Erro na palavra "atualization"
actualization, equalization
```

Nem todas as transformações são possíveis com o algoritmo acima. Por exemplo, só é possível apagar letras num único *offset* no meio das palavras sem outras alterações na palavra. Assim, na palavra errada "centterred", não é considerado o caso de 2 letras duplicadas serem 2 modificações, obtendo o algoritmo um número elevado de letras trocadas.

2.4. Modo de funcionamento 3: correção automática de erros

No modo de funcionamento 3, o programa deve copiar o texto de entrada para o ficheiro de saída, corrigindo as palavras erradas em cada linha de texto para a primeira alternativa encontrada, conforme descrito no modo de funcionamento 2.

3. Interface de entrada/saída e parametrização do programa

Por omissão, o programa deve ler os dados do "stdin" e escrever o resultado da execução em "stdout". O programa pode ser parametrizado de forma a alterar este comportamento por omissão.

O programa não deve escrever nada no output para além do pedido, pois qualquer diferença no output em relação ao esperado será considerado um erro ao comparar com o output esperado.

De seguida descrevem-se os parâmetros de linha de comando que deverão ser interpretados pelo programa a desenvolver.

3.1. Parametrização do programa

O programa deverá ser invocado na linha de comando da seguinte forma:

$ ./ortografia [OPTIONS]

'**$**' é a prompt do Linux e "**./**" representa a directoria corrente.
ortografia designa o nome do ficheiro executável contendo o programa desenvolvido.

[OPTIONS] designa a possibilidade de o programa ser invocado com diferentes opções de funcionamento

As **opções** de funcionamento são identificadas sempre como *strings* começadas com o caractere '-', e podem aparecer por qualquer ordem. De seguida, descrevem-se as várias opções disponíveis:

-h	mostra a ajuda para o utilizador e termina
-i filename	nome do ficheiro de entrada, em alternativa a *stdin*
-o filename	nome do ficheiro de saída, em alternativa a *stdout*
-d filename	nome do ficheiro de dicionário a usar
-a *nn*	o número máximo de alternativas a mostrar com cada erro ortográfico deve ser *nn*
-n *nn*	o número máximo de diferenças a considerar nos erros ortográficos deve ser *nn*
-m *nn*	o modo de funcionamento do programa deve ser *nn*

- A opção -h, quando invocada, deverá imprimir para *stdout* uma mensagem de ajuda de execução da linha de comandos. Para ver um exemplo deste tipo de mensagens, executar no terminal: "vi -h". Também pode apresentar esta mensagem de ajuda no caso haver algum parâmetro inválido.
- Para a opção -d, o ficheiro de dicionário por omissão a usar deve ser "words".
- Para a opção -a, o valor por omissão deve ser 10.
- Para a opção -n, o valor por omissão deve ser 2.
- Para a opção -m, o valor por omissão deve ser 1.

Pode ignorar opções em modos de funcionamento que não as usem. Por exemplo, a opção -a pode ser ignorada no modo de funcionamento 1.

3.2. Exit code na Invocação do Programa

O programa deve sair com um exit code de EXIT_SUCCESS (0) numa execução bem sucedida, ou com um exit code de EXIT_FAILURE (1) numa execução com erro.

3.3. Exemplos de Invocação do Programa

De seguida apresentam-se alguns exemplos válidos e inválidos de invocação do programa com especificação de parâmetros em linha de comando.

Exemplos válidos de invocação do programa:

Exemplo 1: ./ortografia -h
Exemplo 2: ./ortografia -a 5 -m 2
Exemplo 3: ./ortografia -m 2 -d meu_dicionario.txt
Exemplo 4: ./ortografia -m 3 -i 1paragraph.txt -o 1paragraph-output.txt

Exemplos inválidos de invocação do programa:

Exemplo 1: ./ortografia -a -3
Exemplo 2: ./ortografia -m xpto
Exemplo 3: ./ortografia -a 8 -s 0

Nota: Pode recorrer à função de biblioteca getopt() para efetuar a validação dos parâmetros de entrada ou implementar uma função para realizar essa validação. Consulte a página do manual com o comando "man 3 getopt" num terminal Linux, ou numa pesquisa no Google.

No caso de invocação inválida do programa, o programa deve escrever uma mensagem de erro no output e terminar com exit code igual a EXIT_FAILURE (1).

4. Processo de Submissão

Os trabalhos submetidos irão ser **compilados com as seguintes opções**:

-Wall -O3 -g

O executável deve correr na máquina virtual fornecida, onde serão realizados testes.

O projeto deve ser submetido num **ficheiro ZIP** com:

(1) Código comentado com as funcionalidades indicadas no enunciado (ficheiros .h e .c);
(2) Makefile para gerar o executável;

Para além disso, será necessário preencher uma **ficha de auto-avaliação no GoogleForms**.

Por fim, o código deve ser estruturado de forma lógica em vários ficheiros (*.c e *.h). As funções devem ter um cabeçalho curto, mas explicativo, e o código deve estar corretamente indentado e com comentários que facilitem a sua legibilidade.

É importante reforçar que certos modos de operação do programa serão avaliados de forma automática, pelo que é imperativo que o programa respeite a impressão para *stdout*/ficheiro, e a leitura de *stdin*/ficheiro. A falha na execução dos mesmos levará a uma penalização na nota final.

5. Processo de Avaliação

O projeto será avaliado em 2 fases. Na primeira fase, o programa só precisa de ler dados do *stdin* e escrever o resultado no *stdout*, implementando-se apenas o modo de funcionamento 1. Na segunda fase deverão estar operacionais todas as funcionalidades descritas no enunciado. A não utilização de memória dinâmica na fase 2 funciona como uma penalização, caso não seja usada.

Fase	Implementação	Cotação [%]	Cotação [valores]
1	Modo de funcionamento 1: com *stdin*/*stdout*	25	5
2	Modo de funcionamento 2	40	8
2	Modo de funcionamento 3	10	2
2	Parâmetros da linha de comando	5	1
2	Verificação de erros com Valgrind	10	2
2	Qualidade do código: comentários, indentação, estruturação em funções, sem *warnings*, nomes de variáveis apropriados, etc.	10	2
2	Não utilização de memória dinâmica (penalização)	-20	-4

Avaliação oral (**realizada após a entrega do projeto**):

- Grupos com **nota superior a 17** realizarão obrigatoriamente oral para defender a nota (a nota atribuída antes da oral será o ponto de partida e limite máximo da nota após a discussão oral)
- Grupos com **nota entre 8 e 17** só realizarão oral se o docente de laboratório ou o avaliador de projeto considerar necessário.
- Grupos com **nota inferior a 8** podem fazer oral para obter nota mínima de projeto.

6. Código de Honestidade Académica

Espera-se que os alunos conheçam e respeitem o Código de Honestidade Académica que rege esta disciplina, e que pode ser consultado na página da cadeira.

O projeto é para ser planeado e executado por grupos de dois alunos, e é nessa base que será avaliado. Espera-se que ambos os elementos do grupo conheçam em detalhe a solução apresentada para o projeto, sabendo explicar o funcionamento de todo o código que for entregue.

Deverá ser entregue código realizado pelos elementos que compõem o grupo, sem partes provenientes de outros grupos de trabalho, ou de ferramentas automáticas (e.g., baseadas em inteligência artificial). Quaisquer associações de grupos ou outras, que eventualmente venham a ocorrer, serão interpretadas como violação do Código de Honestidade Académica, e terão como consequência a anulação do projeto aos elementos envolvidos.

Lembramos igualmente que a verificação de potenciais violações a este código de conduta é feita de forma automática com recurso a métodos de comparação de código, que envolvem não apenas a comparação direta do código, mas também da estrutura do mesmo.