Instituto Superior de Engenharia de Lisboa

LEETC

Programação II

2021/22 – 1.º semestre letivo

Terceira Série de Exercícios

Pretende-se uma nova implementação do programa especificado na segunda série de exercícios, com as modificações seguintes:

- Exploração de alojamento dinâmico de memória;
- Organização do código em módulos separados.

O alojamento dinâmico é utilizado para conter os *arrays* utilizados em diversas situações, em vez da sua declaração direta como variáveis de dimensão fixa.

A organização em módulos visa estruturação independente de funcionalidades específicas como, por exemplo, a leitura do ficheiro, a separação das palavras ou a gestão dos dados armazenados. As funções anteriormente desenvolvidas e que continuem a ser úteis devem ser reutilizadas, integradas na organização em módulos referida.

Tal como especificado na série de exercícios anterior, o programa realiza a contagem de ocorrências de palavras num ficheiro de texto codificados em UTF-8, armazenando a respetiva informação e suportando comandos de listagem e pesquisa por palavra ou por quantidade relativa de ocorrências. Tem igualmente a operação em duas fases, processamento das palavras e interação com o utilizador. Reproduz-se a especificação dos comandos:

- `a` (*alphabetic*) Apresenta a lista de todas as palavras, por ordem alfabética crescente, e a respetiva frequência;
- `w` *`word`* (*word frequency*) Apresenta a frequência da palavra indicada por *`word`*; no caso de não existir, apresenta a informação de insucesso;
- `+` *`number`* (*most frequent words*) Apresenta um subconjunto das palavras mais frequentes, por ordem decrescente de frequência; o parâmetro *`number`* indica a quantidade de palavras;
- `-` *`number`* (*least frequent words*) Apresenta um subconjunto das palavras menos frequentes, por ordem crescente de frequência; o parâmetro *`number`* indica a quantidade de palavras;
- `q` (*quit*) Termina.

1. Módulos propostos

Organize o programa em módulos, caraterizando a funcionalidade e definindo as assinaturas das funções de interface de cada módulo. Propõe-se que considere os módulos indicados abaixo; no caso de os alunos terem implementado a série anterior em módulos, podem manter a respetiva organização, desde que seja adequada para implementar a aplicação pretendida e não implique a integração de código desnecessário no executável.

1.1. Módulo utilitário para separação de palavras, disponibilizando a interface para iniciar a separação das palavras, tendo como parâmetro a linha, e obter, uma a uma, as palavras que a linha contém.

1.2. Módulo utilitário para normalização de palavras, disponibilizando a interface para normalizar as palavras conforme a especificação da série anterior.

1.3. Módulo utilitário de gestão dos dados, disponibilizando funções para:

Criar as estruturas de dados;
Adicionar cada palavra, fazendo a respetiva contagem;
Terminar a fase de inserção, preparando os dados para as pesquisas;
Realizar as pesquisas especificadas;
Destruir as estruturas de dados, libertando a memória ocupada.

1.4. Módulo de aplicação final, para implementação do programa `wfr` (*word frequency report*) responsável por gerir a obtenção e armazenamento dos dados, bem como a interação com o utilizador, utilizando as funcionalidades dos módulos anteriores para:

Iniciar o funcionamento da gestão dos dados;
Aceder ao ficheiro, indicado em parâmetro de linha de comando, obter a respetivas linhas, separá-las em palavras e transmiti-las ao módulo de gestão dos dados;
Acionar a mudança da fase de processamento para a fase de interação;
Aceitar os comandos do utilizador e promover as pesquisas correspondentes;
Finalizar a atividade de forma ordenada, promovendo nomeadamente a libertação da memória alojada dinamicamente, antes de terminar a execução.

2. Desenvolvimento

2.1. Escreva o módulo utilitário para separação de palavras.

Escreva o código e o respetivo *header file*, contendo as declarações necessárias para a utilização do módulo, constando nomeadamente as assinaturas das funções de interface; no caso de definir funções auxiliares, estas devem ter *scope* privado.

Verifique o funcionamento do módulo, criando uma aplicação de teste e o correspondente *makefile*.

2.2. Escreva o módulo utilitário para normalização de palavras.

Escreva o código e o respetivo *header file*, contendo as declarações necessárias para a utilização do módulo, constando nomeadamente as assinaturas das funções de interface; no caso de definir funções auxiliares, estas devem ter *scope* privado.

Verifique o funcionamento do módulo, criando uma aplicação de teste e o correspondente *makefile*.

2.3. Desenvolva o módulo utilitário de gestão de dados

Defina a estrutura de dados para este módulo, utilizando a nova versão do tipo `WordFreq`:

```
typedef struct{
    char *word;      // aponta palavra armazenada em memória alojada dinamicamente
    int freq;        // número de ocorrências
} WordFreq;
```

Com vista ao uso racional da memória, na nova definição de `WordFreq`, o espaço para as *strings* que representam as palavras, apontado pelos respetivos campos `word`, é alojado dinamicamente, de acordo com a dimensão efetiva de cada palavra.

Propõe-se, nesta versão, que os elementos do tipo `WordFreq` sejam alojados um a um, à medida que as palavras são adicionadas.

Para suportar o acesso às palavras com os dois critérios de ordenação (alfabético para os comandos "`a`" e "`w`"; por frequência para os comandos "`+`" e "`-`") são agora usados dois *arrays* de ponteiros idênticos, ambos alojados dinamicamente. Cada elemento destes *arrays* aponta para uma estrutura `WordFreq` alojada individualmente. Cada estrutura `WordFreq` é apontada por dois ponteiros, cada um pertencente a um dos *arrays*. A dimensão dos *arrays* de ponteiros deve adaptar-se automaticamente à quantidade de palavras, com recurso à função `realloc` da biblioteca normalizada. O redimensionamento deve realizar-se por blocos, de modo a reduzir o *overhead* de processamento na execução de `realloc`.

A ordenação e a pesquisa nos *arrays* de ponteiros referidos devem ser realizadas, com os respetivos critérios, utilizando as funções `qsort` e `bsearch` da biblioteca normalizada. As ordenações devem ser asseguradas no final da fase de processamento, aquando da transição para a fase de interação.

O *array* de ponteiros que suporta o acesso alfabético pode não necessitar de ser ordenado no final se estiver sempre ordenado, de forma semelhante à especificada no exercício 3 (opcional) da série anterior.

O *array* de ponteiros que suporta o acesso por frequência pode ser criado ao longo da inserção das palavras, tal como o anterior, ou apenas no final. Neste caso, beneficia de já se conhecer a quantidade de palavras, evitando a necessidade de usar `realloc`. Em qualquer dos casos, a ordenação deste *array* tem que ser realizada no final, após concluir a inserção de todas as palavras, de modo que as respetivas frequências estejam identificadas.

Escreva o código do módulo e o respetivo *header file*, contendo as declarações necessárias para a sua utilização. Deve conter, nomeadamente, as assinaturas das funções de interface. No caso de definir variáveis permanentes para a gestão do módulo ou funções auxiliares, estas devem ter *scope* privado.

Verifique o funcionamento do módulo, criando uma aplicação de teste e o correspondente *makefile*.

2.4. Desenvolva o módulo de aplicação final

Rescreva o código da aplicação final, adaptado a implementação anterior de modo a usar as funções de interface dos outros módulos.

Nesta versão da aplicação deve utilizar a função de biblioteca `getline`, em substituição de `fgets`, para realizar a leitura do ficheiro. Isto permite o alojamento dinâmico e dimensionamento automático da memória para a linha de texto. Recomenda-se a leitura da especificação obtida, na linha de comando, com "`man getline`".

Escreva o *makefile* para controlar, de forma eficiente, a produção do executável.

Teste o programa completo, reutilizando os ficheiros de texto usados para ensaio da série anterior; pode criar outros ficheiros se considerar conveniente.

ISEL, 2021.12.06