Instituto Superior de Engenharia de Lisboa

LEETC

Programação II

2024/25 – 1.º semestre letivo

Terceira Série de Exercícios

Pretende-se uma nova implementação do programa especificado na Série de Exercícios 2 (SE2), explorando os mecanismos de alojamento dinâmico, para flexibilizar o dimensionamento dos dados a processar.

Tal como na versão anterior, a informação relativa a uma lista de livros é obtida de um ficheiro de texto e depositada numa estrutura de dados, agora baseada em alojamento dinâmico. Após esta fase, o programa deve esperar, em ciclo, os comandos "l", "i", "a" e "q" especificados na SE2. No caso do comando "a", a nova implementação recorre a uma estrutura de dados adicional, para melhorar a eficiência das pesquisas.

A modificação da estrutura de dados anteriormente existente consiste em usar alojamento dinâmico de memória, de modo a permitir o dimensionamento flexível, em função da quantidade dados recebidos:

- Os elementos de informação essenciais são representados pelo tipo *struct* `Book`, semelhante ao antigo tipo `BookData`, com a diferença de alojar dinamicamente as *strings* que podem ter dimensão diversa. Para isso, os respetivos campos passam a ser ponteiros para espaço alojado dinamicamente com a dimensão efetivamente necessária para as *strings*. Excetua-se o campo `isbn` que é um *array* de `char`, dado que esta *string* tem dimensão fixa. Os elementos do tipo `Book` são alojados dinamicamente, um a um em blocos individuais, e referenciados através de outras estruturas de dados, também dinâmicas;
- As estruturas para referenciar os elementos `Book` são *arrays* de ponteiros com alojamento dinâmico, designados por vetores, cujo redimensionamento é realizado pela função `realloc` de biblioteca. Propõe-se a utilização de dois vetores destes, para referenciar os elementos de forma independente com duas ordenações específicas para os comandos "l" e "i".

Os tipos de *struct* propostos para construir a estrutura de dados são os seguintes:

```
typedef struct {      // Descritor dos dados de um livro
    char *title;      //  string alojada dinamicamente
    char isbn[MAX_ISBN]; // string com dimensão fixa
    char *authors;    //  string alojada dinamicamente
    char *publisher; //  string alojada dinamicamente
} Book;

typedef struct{       // Descritor de um vetor
    Book **refs;      //  array alojado dinamicamente
    int size;         //  quantidade de elementos preenchidos
    int space;        //  quantidade de elementos alojados
} VecBookRef;

typedef struct{                 // Descritor de um vetor
    VecBookRef *titleVec;       //  vetor a ordenar por título
    VecBookRef *isbnVec;        //  vetor a ordenar por isbn
} DynCollection;
```

Ao construir as estruturas de dados, cada elemento `Book` será alojado e preenchido com os dados provenientes do ficheiro, sendo de seguida o seu endereço adicionado aos *arrays* dinâmicos para referenciação.

Após o preenchimento, realiza-se a ordenação dos vetores de referenciação, um alfabeticamente por título, para responder ao comando "l" em percurso sequencial, e o outro por ISBN, para responder ao comando "i" em pesquisa binária.

No desenvolvimento do programa, devem ser reutilizados os módulos e *header files* anteriormente criados, se adequados, ou parte do seu conteúdo, adaptado para desenvolver outros.

Propõe-se o desenvolvimento organizado nas fases seguintes, escrevendo e testando sucessivamente cada uma das funcionalidades especificadas.

1. Construção e preenchimento da estrutura de dados dinâmica.

   Transcreva a declaraçõ es e definições relativas à estrutura de dados proposta. Escreva o *software* especificado abaixo, desenvolvendo os módulos de código fonte e os respetivos *header files*.

1.1. Escreva o código para gerir os descritores de livro, `bookCreate` e `bookFree`.

```
Book *bookCreate( const char *line );
```

destina-se a criar o descritor de um livro com os campos obtidos de uma linha de texto indicada por `line`. Deve alojar dinamicamente o espaço necessário para uma estrutura do tipo `Book`, bem como para as *strings* a associar aos campos `title`, `authors` e `publisher`. Esta função retorna o endereço do `Book` alojado, em caso de sucesso, ou NULL, se alinha de texto não é válida.

Esta função pode ser construída por adaptação da antiga função `fillBookData`. Deve ter o cuidado de não deixar espaço alojado sem utilização, nomeadamente quando a linha é inválida.

```
void bookFree( Book *b );
```

liberta a memória de alojamento dinâmico ocupada pela *struct* `Book` indicada por `b` e pelas *strings* dela dependentes.

1.2. Escreva o código para gerir os vetores de referências, com as funções `vecRefCreate`, `vecRefAdd`, `vecRefSize`, `vecRefGet`, `vecRefSortTitle`, `vecRefSortIsbn`, `vecRefSearchIsbn` e `vecRefFree`.

```
VecBookRef *vecRefCreate ( void );
```

aloja dinamicamente o espaço para o descritor de um vetor de referências e preenche-o com as condições iniciais. O campo `size` deve ter o valor 0, indicando o estado vazio; O campo `space` deve ter o valor coerente com a iniciação do campo `refs`.

```
void vecRefAdd( VecBookRef *vec, Book *ref );
```

adiciona a referência de um livro. O parâmetro `vec` indica o descritor do vetor. O parâmetro `ref` indica o livro a referenciar. O espaço alojado para o *array* de ponteiros deve ser redimensionado quando necessário, utilizando a função `realloc` de biblioteca. O redimensionamento deve ocorrer em blocos de vários elementos, por motivos de eficiência.

```
void vecRefSize( VecBookRef *vec );
```

retorna o número de elementos referenciados no vetor de referências identificado por `vec`.

```
Book *vecRefGet( VecBookRef *vec, int index );
```

retorna o endereço elemento referenciado na posição `index` do vetor de referências identificado por `vec`. Se a posição indicada for inválida, retorna NULL.

```
void vecRefSortTitle( VecBookRef *vec );
```

ordena o *array* de referências identificado por `vec`, alfabeticamente crescente pelo título dos livros referenciados, para suportar o comando “l”. Deve usar a função `qsort` de biblioteca.

```
void vecRefSortIsbn( VecBookRef *vec );
```

ordena o vetor de referências identificado por `vec`, alfabeticamente crescente pelo ISBN dos livros referenciados, para suportar o comando "i". Deve usar a função `qsort` de biblioteca.

```
Book *vecRefSearchIsbn( VecBookRef *vec, char *isbn );
```

procura, no vetor de referências identificado por `vec`, o livro com o ISBN indicado. Retorna a referência do livro ou NULL, se não existir. Deve usar a função `bsearch` de biblioteca.

```
void vecRefFree( VecBookRef *vec, int freeBooks );
```

liberta o espaço de alojamento dinâmico usado pelo descritor indicado por `vec` e pelo respetivo *array* de referências.

Se o valor de `freeBooks` for *true*, liberta também o espaço usado pelos descritores de livro. Note que, por haver dois vetores a referenciar os mesmos livros, esta função não pode libertar os livros sempre que elimina um vetor.

1.3. Escreva o código para gerir a coleção de livros, com as funções `dynCollCreate`, `dynCollAddBook`, `dynCollFill` e `dynCollFree`.

```
DynCollection *dynCollCreate( void );
```

destina-se a criar o descritor da coleção alojada dinamicamente, no estado vazio.

```
int dynCollAddBook( const char *line, void *context );
```

destinada a ser passada no parâmetro `action` da função `processFile` (desenvolvida na SE2) para adicionar os dados de um livro ao armazenamento de uma coleção, referenciando nos dois vetores.

A linha de texto indicada por `line` contém os campos de informação, na forma obtida do ficheiro. O parâmetro `context` representa o endereço do descritor com tipo `DynCollection` que armazena as referências para os dados. Deve utilizar a função `bookCreate`, para criar a representação do novo elemento. Esta função retorna 1, em caso de sucesso, ou 0, no caso contrário, devido a linha incorreta.

```
void dynCollFill( DynCollection *coll, FILE *f );
```

destina-se a preencher o descritor da coleção, alojada dinamicamente, com os dados provenientes do ficheiro `f`, previamente aberto. Deve usar a função `processFile`, passando a função `dynCollAddBook`, para preencher a coleção. No final, deve ordenar os dois vetores de referências pelos respetivos critérios.

```
void dynCollFree( DynCollection *coll );
```

liberta o espaço alojado dinamicamente para o descritor da coleção indicado por `coll` e todas as estruturas dele dependentes. Ao eliminar os dois vetores de referências, deve configurar o parâmetro `freeBooks` de `vecRefFree` para promover a eliminação dos livros apenas uma vez.

2. Implementação dos comandos "l" e "i"

2.1. Prepare o *makefile* envolvendo os módulos acima especificados e o módulo de aplicação, de modo a realizar a compilação eficiente e a respetiva produção do executável.

Escreva e teste o módulo de aplicação capaz de responder aos comandos "l" e "i", ou adapte o realizado na SE2, explorando as estruturas de dados dinâmicas. Para construir e utilizar as estruturas de dados, deve usar funções das especificadas nos pontos anteriores. Pode escrever as funções auxiliares que considerar convenientes.

O comando "l" explora o vetor de referências ordenado por título, pertencente ao descritor da coleção, usando as funções `vecRefSize` e `vecRefGet`.

O comando "i" explora o vetor de referências ordenado por ISBN, realizando a pesquisa binária através da função `vecRefSearchIsbn`.

A este programa serão, adiante, adicionados os comandos "a" e "q". Não se pretende que conserve o executável desta fase de desenvolvimento, dado que esta funcionalidade integra a versão final.

3. Estrutura de dados para pesquisa eficiente por palavras do nome de autor

Com vista a uma implementação eficiente para o comando "a", anteriormente especificado na SE2, é proposta ema estrutura de dados dinâmica, explorando os modelos de árvore binária de pesquisa e de lista ligada.

A estrutura proposta armazena todas as palavras existentes em nomes de autores e estabelece uma relação entre cada uma delas e os livros onde esta existe.

- A componente principal da estrutura de armazenamento é uma árvore binária em que cada nó representa uma das palavras existentes em nomes de autores;
- Em cada nó da árvore é criada uma lista ligada de referências, as quais identificam os livros que contêm, no nome dos autores, a palavra indicada.

Pretende-se a criação de dois novos módulos e respetivos *header files*, de modo a implementar, em separado, o *software* de listas ligadas e de árvore binária de pesquisa.

3.1. Prepare a utilização do tipo `LNode`, destinado a implementar listas ligadas de referências

```
typedef struct lNode{
      struct lNode *next;   // ligação na lista
      Book *ref; // referência de acesso a um descritor de livro
} LNode;
```

3.2. Escreva o módulo de gestão de listas ligadas, com as funções `lRefAdd`, `lRefPrint` e `lRefFree`.

```
int lRefAdd( LNode **headPtr, Book *ref );
```

adiciona à lista ligada um nó com a referência para um livro. O parâmetro `headPtr` identifica o endereço do ponteiro cabeça da lista, o qual poderá ser modificado pela função. O parâmetro `ref` indica o livro que se pretende referenciar. A lista deve ficar ordenada crescentemente pelo título dos livros referenciados. A função retorna: 1, em caso de sucesso; 0, no caso de o livro já estar referenciado na lista, a qual neste caso não é modificada.

```
void lRefPrint( LNode *head );
```

percorre sequencialmente a lista identificada por `head` e apresenta os dados dos livros referenciados, conforme especificado para o comando "a".

```
void lRefFree( LNode *head );
```

liberta o espaço de alojamento dinâmico ocupado pelos nós da lista indicada por `head`. Assume-se que não compete a esta função libertar o espaço ocupado pelos livros referenciados.

3.3. Prepare a utilização do tipo `TNode`, destinado a implementar listas ligadas de referências

```
typedef struct tNode{
      struct tNode *left, *right;
      char *word;
      LNode *head;
} TNode;
```

3.4. Escreva o módulo de gestão da árvore binária de pesquisa, com as funções `bstAdd`, `bstBalance`, `bstSearch` e `bstFree`.

```
void bstAdd( TNode **rootPtr, char *namWord, Book *ref );
```

adiciona à árvore uma ocorrência da palavra indicada por `namWord`; se a palavra não existir, é inserida. O parâmetro `rootPtr` identifica o endereço do ponteiro raiz da árvore, o qual poderá ser modificado pela função. O parâmetro `ref` indica o livro que se pretende referenciar, associado à palavra `namWord`. A árvore é ordenada alfabeticamente. A inserção é sempre realizada nas folhas, com o propósito de simplificar o algoritmo.

```
void bstBalance( TNode **rootPtr );
```

reorganiza a árvore de modo que fique balanceada. Esta função destina-se a ser executada após as inserções, de modo a melhorar a eficiência das pesquisas, já que a inserção simples nas folhas pode produzir árvores desbalanceadas.

```
LNode *bstSearch ( TNode *root, char *namWord );
```

procura, na árvore identificada pela raiz `root`, a palavra `namWord`. Retorna o endereço do primeiro elemento da lista de referências para os livros que contêm essa palavra no nome dos autores. No caso de a palavra não existir, retorna NULL.

```
void bstFree( TNode *root );
```

liberta o espaço de alojamento dinâmico ocupado pelos nós da árvore e por outros elementos que deles dependam.

4. Implementação do comando “a” na aplicação

É necessário desenvolver as partes do programa que utilizem a estrutura de dados proposta, de modo a inserir as referências e a utilizá-las posteriormente.
O código a desenvolver terá componentes que executam, pelo menos, nas fases seguintes:

- No final da fase de leitura da informação, após o preenchimento dos vetores de referências, construir a árvore binária e as listas ligadas a ela associadas, para referenciação dos livros;
- Após a construção da árvore, aplicar-lhe o balanceamento;
- No ciclo de interpretação dos comandos, executar as pesquisas em resposta ao comando “a”.

Desenvolva o código para construir a estrutura de dados. Note que é necessário processar os campos de autor de cada um dos livros, identificando as palavras isoladamente e adicionando à árvore binária a ocorrência de cada palavra, associada ao livro a que pertence. Note que o código de inserção na árvore realiza a inserção da referência par o livro numa lista ligada anexa à palavra.

Para isolar as palavras, sugere-se que adapte o código com essa funcionalidade, realizado na SE2.

Para adicionar à árvore binária as palavras e as respetivas referências, deve explorar um vetor de referências pertencente ao descritor da coleção, usando as funções `vecRefSize` e `vecRefGet`.

O critério de ordenação das listas ligadas deve ter em conta que o comando “a” apresenta os resultados ordenados por título. Para construir estrutura de dados de forma eficiente, propõe-se que aceda ao vetor ordenado por título, percorrendo-o de forma a favorecer a ordenação pretendida para as listas ligadas. Pode escrever, no domínio da aplicação, quaisquer funções que considere convenientes.

5. Libertação da memória alojada dinamicamente

Pretende-se, na finalização da aplicação, libertar de todas as estruturas de dados alojadas dinamicamente. Adicione, no processamento do comando “q”, a chamada às funções para libertar a memória alojada dinamicamente em todas as estruturas de dados.

Anexo – Balanceamento da árvore binária

Para uma utilização eficiente, as árvores binárias devem ser balanceadas. Propõe-se, para simplificar, que as crie sem manter permanentemente o balanceamento, sendo este realizado através da função `bstBalance`, a intervalos de várias inserções ou, pelo menos, no final. O código proposto abaixo considera o nó de árvore com o tipo `TNode` os campos de ligação com os nomes `left` e `right`.

Para implementar a função `tBalance`, propõe-se a técnica de balanceamento em dois passos:

1. Transformar a árvore binária numa árvore degenerada em lista ordenada, ligada pelo campo `right`, usando o algoritmo seguinte.

```
TNode *treeToSortedList( TNode *r, TNode *link ){
    TNode * p;
    if( r == NULL ) return link;
    p = treeToSortedList( r->left, r );
    r->left = NULL;
    r->right = treeToSortedList( r->right, link );
    return p;
}
```

2. Conhecido o número de elementos, transformar a lista numa árvore, usando o algoritmo seguinte.

```
TNode *sortedListToBalancedTree(TNode **listRoot, int n) {
    if( n == 0 )
          return NULL;
    TNode *leftChild = sortedListToBalancedTree(listRoot, n/2);
    TNode *parent = *listRoot;
    parent->left = leftChild;
    *listRoot = (*listRoot)->right;
    parent->right = sortedListToBalancedTree(listRoot, n-(n/2 + 1) );
    return parent;
}
```

ISEL, 2024.11.25