Instituto Superior de Engenharia de Lisboa

Programação II

2022/23 – 2.º semestre letivo

Teste de época normal

2023.06.23

Nota importante: Valoriza-se a escrita de código que inclua comentários esclarecedores da implementação seguida e que contenha indentação legível.

1. [6 valores]

Para a realização da **Série de Exercícios 3** foi utilizado o tipo `BookData` como elemento de uma estrutura de dados para representar os dados de livros, e o tipo `VecBookRef` como descritor de um conjunto de dados bibliográficos (ou seja, como descritor de acesso a um conjunto de livros). Os descritores dos dados de um livro e de um vetor de livros são declarados da seguinte forma:

```
1   typedef struct {        // Descritor dos dados de um livro
2     char *title;          //  string alojada dinamicamente
3     char isbn[MAX_ISBN]; //  string com dimensão fixa
4     char *authors;        //  string alojada dinamicamente
5     char *publisher;      //  string alojada dinamicamente
6   } Book;
7
8   typedef struct{         // Descritor de um vetor
9     Book **refs;          //  array alojado dinamicamente
10    int size;             //  quantidade de elementos preenchidos
11    int space;            //  quantidade de elementos alojados
12  } VecBookRef;
```

No desenvolvimento das funções seguintes deve ter em conta que o vetor de livros, passado no parâmetro `vec`, está sempre ordenado e deve manter-se ordenado. A ordem é crescente por `isbn`, e o vetor não contém repetições. Na implementação das funções pretendidas, pode usar outras das especificadas, bem como escrever funções auxiliares.

a) [2] Pretende-se o desenvolvimento da função

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

que procura, num vetor de livros, um livro com o **isbn** passado por parâmetro. A função retorna uma referência para o livro com o **isbn** passado por parâmetro, caso exista, e retorna `NULL` caso não seja encontrado qualquer livro.

b) [2] Pretende-se o desenvolvimento da função

```
int delete_book(VecBookRef *vec, char *isbn);
```

que remove de um vetor de livros a referência para o livro cujo **isbn** é passado por parâmetro. A função mantém o vetor ordenado e retorna: 0 se nenhum livro foi removido (porque não existia qualquer livro com aquele **isbn**); um valor diferente de 0 caso tenha havido remoção. A função delete_book não elimina o descritor de livro referenciado pela posição eliminada no vetor.

c) [2] Pretende-se o desenvolvimento da função

```
int insert_book(VecBookRef *vec, Book *b);
```

que se destina a inserir no vetor, indicado por `vec`, um novo elemento para referenciar o livro indicado por `b`, mantendo o vetor ordenado e sem repetições. Se já existir um elemento igual, não se realiza qualquer modificação. A função deve assegurar espaço no vetor, alojado dinamicamente, para o novo elemento; se necessário, deve redimensionar o espaço, fazendo-o em blocos de 10 elementos, com recurso à função `realloc` de biblioteca.

(esta alínea continua na página seguinte)

A função `insert_book` retorna: 0, caso o livro referenciado por `b` já exista no vetor; 1, em caso de sucesso na inserção; -1, em caso de insucesso por falta de memória de alojamento dinâmico. Note que, caso não haja memória dinâmica disponível, o vetor deve ser mantido inalterado.

Sugere-se que percorra o vetor sequencialmente para identificar o ponto de inserção do novo elemento. No exercício desta alínea não deve utilizar a função `qsort` nem a função `bsearch` da biblioteca normalizada.

2. [5 valores]

Pretende-se implementar uma aplicação de apoio a filas de espera, para organizar a chamada de pessoas que aguardam atendimento num serviço. A estrutura de dados principal é uma lista ligada formada por nós do tipo `Waiter`. O acesso à lista ligada faz-se através de um descritor com o tipo `Queue`, cujos campos devem ser definidos pelo aluno.

```
typedef struct waiter {
    struct waiter *next;   // ligação em lista
    char *name;            // nome do utente; alojar dinamicamente
} Waiter;

typedef struct {
    /*
    ... (campos a definir pelo aluno)
    */
} Queue;
```

A fila deve suportar as operações de colocar elementos e de os obter de volta (retirando da estrutura de dados), assegurando que se mantenham por ordem e que o elemento retirado seja sempre o mais antigo.

a) [1] Escreva a definição completa do tipo `Queue`, descritor de fila, de modo a facilitar a implementação o mais eficiente possível das operações de colocar e obter (retirando) elementos.

b) [1,5] Escreva a função

```
void qPut( Queue *q, char *sn );
```

que coloca na fila `q` um novo elemento, associando ao respetivo campo `name` uma cópia, alojada dinamicamente, da *string* `sn`. Assuma, por simplificação, que nunca ocorre falta de memória para alojamento dinâmico.

c) [1,5] Escreva a função

```
int qGet( Queue *q, char *dn );
```

destinada a obter, retirando da fila, o elemento mais antigo. A função retorna: 1, em caso de sucesso; 0, se a fila estiver vazia.

Quando retira um elemento, deve copiar o nome que ele contém para o espaço indicado por `dn` e libertar toda a memória de alojamento dinâmico usada no elemento. Assuma, por simplificação, que o espaço apontado por `dn` é suficiente para depositar o nome.

d) [1] Escreva a função

```
int qCount( Queue *q );
```

que retorna a quantidade de elementos existentes na fila.

3. [5 valores]

Pretende-se implementar conjuntos de valores numéricos de vírgula flutuante, na forma de árvores binárias de pesquisa. Os elementos dos conjuntos são representados pelo tipo `SetNode`.

```
typedef struct setNode {
    struct setNode *less, *great; // ligações na árvore
    double value;                 // número representado
} SetNode;
```

As árvores binárias que representam os conjuntos são ordenadas pelo valor dos elementos, com os menores ligados pelo campo `less`. De acordo com o conceito matemático de conjunto, não há valores repetidos.

Na implementação das funções pretendidas, pode usar outras das especificadas, bem como escrever funções auxiliares. Valoriza-se a eficiência.

a) [2] Escreva a função

```
SetNode *setInsert( SetNode *s, double v );
```

que insere um elemento com o valor `v` (se não existir) no conjunto representado por `s`. A função retorna o ponteiro de acesso ao conjunto, eventualmente modificado pela inserção.

b) [1,5] Escreva a função

```
int setContains( SetNode *s, double v );
```

que procura o elemento com o valor `v` no conjunto representado por `s`. A função retorna 1 se o valor `v` existe ou 0 no caso oposto.

c) [1,5] Escreva a função

```
SetNode *setIntersect( SetNode *s1, SetNode *s2 );
```

que cria um novo conjunto formado pelos valores que existem em ambos os conjuntos originais, indicados por `s1` e `s2`. Retorna o ponteiro de acesso ao novo conjunto. Não modifica os conjuntos originais.

4. [4 valores]

Considere o conjunto de funções implementadas nos módulos utilitários, `m1.c`, `m2.c` e `m3.c`, seguintes:

m1.c	m2.c	m3.c

`static int faux(int x){`
`  return f2( x );`
`}`
 	
`int f1( int x ){`
`  return faux( x );`
`}`
	`static void faux(int x){`
	`  printf( "%d\n", x );`
	`}`
	
	`int f2( int x ){`
	`  faux( x );`
	`  return x;`
	`}`
		`int f3(int x, int y){`
		`  f2( x );`
		`  return y;`
		`}`

a) [1] Relativamente a cada um dos módulos compilados correspondentes, `m1.o`, `m2.o` e `m3.o`, apresente as listas com os símbolos e a respetiva classificação (**`T`**, *public text*; **`t`**, *private text*; **`U`**, *undefined*.).
Nota que nome de função `faux`, existe em dois módulos, `m1.c` e `m2.c`. Diga o se é possível integrar o código destes dois módulos, simultaneamente, no executável de uma aplicação; justifique.

b) [1] Admita que são usados vários *header files*, `m1.h`, `m2.h` e `m3.h`, para representar informação relativa, respetivamente, aos módulos `m1.c`, `m2.c` e `m3.c`. Escreva o *header file* `m1.h`, contendo as declarações adequadas e o controlo da inclusão múltipla.

c) [1] Indique quais os *header files* que cada módulo deve incluir. Complete o módulo `m3.c` com as diretivas de inclusão e apresente os motivos para incluir, neste módulo, cada um dos *header files* que considerar.

d) [1] Suponha um módulo de aplicação, `myaplic.c`, que, das funções referidas, utiliza diretamente apenas `f1`. Indique, justificando, dos *header files* `m1.h`, `m2.h` e `m3.h`, quais devem ser incluídos por este módulo de aplicação. Escreva um *makefile* para gerar, de forma eficiente, o executável da aplicação com o nome `myaplic`, usando os módulos estritamente necessários.